DESIGN OF A COMPUTER
THE CONTROL DATA 6600

J. E. THORNTON

CENTRAL PROCESSOR INSTRUCTION EXECUTION TIMES

(Times Listed in Minor Cycles)

BRANCH UNIT

00	STOP		—
01	RETURN JUMP to K		14
02	GO TO K + Bi (Note 1)		14
030	GO TO K if Xj = zero	Note 2	9*
031	GO TO K if Xj ≠ zero		9*
032	GO TO K if Xj = positive		9*
033	GO TO K if Xj = negative		9*
034	GO TO K if Xj is in range		9*
035	GO TO K if Xj is out of range		9*
036	GO TO K if Xj is definite		9*
037	GO TO K if Xj is indefinite		9*
04	GO TO K if Bi = Bj	Note 1	8*
05	GO TO K if Bi ≠ Bj		8*
06	GO TO K if Bi ≧ Bj		8*
07	GO TO K if Bi < Bj		8*

Note 1. GO TO K + Bi and GO TO K if Bi . . .
tests made in increment unit
Note 2. GO TO K if Xj . . . tests made in long add unit

* Add 6 minor cycles to branch time for a branch to an instruction which is out of the stack (no memory conflict considered)

BOOLEAN UNIT

10	TRANSMIT Xj to Xi	3
11	LOGICAL PRODUCT of Xj and Xk to Xi	3
12	LOGICAL SUM of Xj and Xk to Xi	3
13	LOGICAL DIFFERENCE of Xj and Xk to Xi	3
14	TRANSMIT Xk COMP. to Xi	3
15	LOGICAL PRODUCT of Xj and Xk COMP to Xi	3
16	LOGICAL SUM of Xj and Xk COMP to Xi	3
17	LOGICAL DIFFERENCE of Xj and Xk COMP to Xi	3

SHIFT UNIT

20	SHIFT Xi LEFT jk places	3
21	SHIFT Xi RIGHT jk places	3
22	SHIFT Xi NOMINALLY LEFT Bj places	3
23	SHIFT Xi NOMINALLY RIGHT Bj places	3
24	NORMALIZE Xk in Xi and Bj	4
25	ROUND AND NORMALIZE Xk in Xi and Bj	4
26	UNPACK Xk to Xi and Bj	3
27	PACK Xi from Xk and Bj	3
43	FORM jk MASK in Xi	3

ADD UNIT

30	FLOATING SUM of Xj and Xk to Xi	4
31	FLOATING DIFFERENCE of Xj and Xk to Xi	4
32	FLOATING DP SUM of Xj and Xk to Xi	4
33	FLOATING DP DIFFERENCE of Xj and Xk to Xi	4
34	ROUND FLOATING SUM of Xj and Xk to Xi	4
35	ROUND FLOATING DIFFERENCE of Xj and Xk to Xi	4

LONG ADD UNIT

36	INTEGER SUM of Xj and Xk to Xi	3
37	INTEGER DIFFERENCE of Xj and Xk to Xi	3

DIVIDE UNIT

44	FLOATING DIVIDE Xj by Xk to Xi	29
45	ROUND FLOATING DIVIDE Xj by Xk to Xi	29
46	PASS	—
47	SUM of 1's in Xk to Xi	8

MULTIPLY UNIT*

40	FLOATING PRODUCT of Xj and Xk to Xi	10
41	ROUND FLOATING PRODUCT of Xj and Xk to Xi	10
42	FLOATING DP PRODUCT of Xj and Xk to Xi	10

INCREMENT UNIT*

50	SUM of Aj and K to Ai	3
51	SUM of Bj and K to Ai	3
52	SUM of Xj and K to Ai	3
53	SUM of Xj and Bk to Ai	3
54	SUM of Aj and Bk to Ai	3
55	DIFFERENCE of Aj and Bk to Ai	3
56	SUM of Bj and Bk to Zi	3
57	DIFFERENCE of Bj and Bk to Zi	3
60	SUM of Aj and K to Bi	3
61	SUM of Bj and K to Bi	3
62	SUM of Xj and K to Bi	3
63	SUM of Xj and Bk to Bi	3
64	SUM of Aj and Bk to Bi	3
65	DIFFERENCE of Aj and Bk to Bi	3
66	SUM of Bj and Bk to Bi	3
67	DIFFERENCE of Bj and Bk to Bi	3
70	SUM of Aj and K to Xi	3
71	SUM of Bj and K to Xi	3
72	SUM of Xj and K to Xi	3
73	SUM of Xj and Bk to Xi	3
74	SUM of Aj and Bk to Xi	3
75	DIFFERENCE of Aj and Bk to Xi	3
76	SUM of Bj and Bk to Xi	3
77	DIFFERENCE of Bj and Bk to Xi	3

*Duplexed units—instruction goes to free unit
Octal Code at left of instruction
Comp—Complement
DP—Double Precision

DESIGN OF A
COMPUTER
THE CONTROL DATA
6600

In the editorial series of
MALCOLM C. HARRISON
Courant Institute of Mathematical Sciences
New York University

"DESIGN OF A COMPUTER THE CONTROL DATA 6600"

J. E. THORNTON

Vice President
Advanced Design Laboratory
Control Data Corporation

SCOTT, FORESMAN AND COMPANY

The type on the cover and title page of *Design of a Computer—The Control Data 6600* is a sample of the 6600 display lettering. The display unit contains two cathode ray tubes and a manual keyboard. Information is displayed in alphabetic and numeric symbols which are formed on the surface of each tube. The symbols are then traced out or "painted" on the phosphor of each CRT by the action of its electron beam. Control of the beam for this purpose is provided by electrostatic deflection in two dimensions, horizontal and vertical. A symbol is painted by electronically converting from the symbol, as it is stored in the computer, to deflection voltages applied to CRT. The letters appearing on the cover of this book were photographed from the display unit.

Library of Congress Catalog Number 74-96462

Copyright © 1970
by Scott, Foresman and Company,
Glenview, Illinois 60025.
Philippines Copyright 1970
by Scott, Foresman and Company.
All Rights Reserved.
Printed in the United States of America.
Regional offices of Scott, Foresman
and Company are located in Atlanta, Dallas, Glenview, Palo Alto,
Oakland, N.J., and London, England.

FOREWORD

In spite of the large number of computing systems which have been designed and are in use today there is no clear-cut optimum approach to a general purpose computing system. Rather, it would seem, we are just beginning to explore the really basic variations from the one address sequential machines that launched the digital computing industry.

Early in digital computer history circuit technology advanced so rapidly that giant strides were made in equipment performance with little variation in design structure. The very presence of this rapid technological advance discouraged exploration of system structure. Electrical circuits tend to interact with system organization, and a good system design could become obsolete in a short period of time because the associated electrical circuits had been passed by.

In the early 1960's electrical circuit performance began to stabilize with the advent of integrated circuit technology. Circuit speed improvement continued but at a somewhat lower rate. In addition the integrated circuit offered the alternative of using larger quantities of mass produced configurations for the same cost as might be obtained by brute force efforts at speed in serial processors.

System design then began to diverge into parallel structures. This book describes one of the early machines attempting to explore parallelism in electrical structure without abandoning the serial structure of the computer programs. Yet to be explored are parallel machines with wholly new programming philosophies in which serial execution of a single program is abandoned.

A book describing the characteristics of a modern large-scale digital computer is a challenging undertaking. There is more detail information to be presented than is possible in a single volume. An overview of the system without being specific is generally too vague to convey the important characteristics that are of real interest. The author in this book selects special areas for detail treatment where those areas are unique to the machine described. These are interconnected with a general description of the system as a whole.

The reader can rest assured that the material presented is accurate and from the best authority as Mr. Thornton was personally responsible for most of the detailed design of the Control Data model 6600 system.

SEYMOUR R. CRAY
Vice President and General Manager
Chippewa Laboratory

TABLE OF CONTENTS

I. INTRODUCTION 1

A. Justification for Large Computers 1
B. Building Blocks 4
C. The Approach 5

II. ORGANIZATION OF THE 6600 9

A. General 9
B. Peripheral Subsystem 10
C. Central Processor—CPU 12
D. Central Storage 15
E. Extended Core Storage 17

III. BASIC CIRCUIT PROPERTIES 19

A. The Silicon Transistor 19
B. DCTL Logic Circuits 21
C. Logic Symbols 24
D. Transmission Lines 28
E. Packaging 32

IV. CENTRAL STORAGE SYSTEM 37

A. Storage Module 37
B. Theory of Interleaved Storage 44
C. Stunt Box 47
D. Storage Bus System 51
E. Extended Core Storage 53
F. ECS Coupler and Controller 55

V. CENTRAL PROCESSOR FUNCTIONAL UNITS 57

- A. Boolean Unit 59
- B. Fixed Add Unit 63
- C. Data Trunks 69
- D. Shift Unit 71
- E. Add Unit 77
- F. Multiply Unit 88
- G. Divide Unit 101
- H. Increment Unit 105
- I. Branch Unit 111
- J. ECS Coupler-Controller 114

VI. CENTRAL PROCESSOR CONTROL 117

- A. Exchange Jump 117
- B. Instruction Fetch 120
- C. Instruction Issue 123
- D. Scoreboard 125
- E. Register Entry/Exit Control 134
- F. Summary 137

VII. PERIPHERAL SUBSYSTEM 141

- A. Peripheral Processors 141
- B. Dead Start 154
- C. Disk Storage 157

VIII. SYSTEMS OPERATION 163

- A. Files 163
- B. Tables 165
- C. Circular Buffer for I/0 166
- D. Job Processing 167

E. System Monitor MTR 168
F. Control Points 169
G. Summary 171

APPENDIX 173

INDEX 177

INTRODUCTION

I

Reduction to practice is a desirable and necessary test of any theory. The growing body of theory and understanding about digital computation is no exception. Particularly evident in recent years are attempts to define new organization or architecture of digital computers which offer significant performance improvement. Of interest are theories involving simultaneous or concurrent computation, sometimes called functional parallelism, subfunction concurrency, multiprocessing, and so on.

This book is offered as a "case study" of a major digital computer which has reduced to practice a number of interesting theories involving parallelism and concurrency.

It is assumed that the reader has been exposed to some introductory study in digital computation and number theory. No attempt is made in the book to establish any all-encompassing theory. While this may be dissatisfying to some, the author feels this is best left to other comparative studies and theoretical works.

Following sections of this chapter are included to provide the background pertinent to the discussion of the 6600 Computer.

A. JUSTIFICATION FOR LARGE COMPUTERS

The motivation for the computer came at least partly from the need to solve systems of linear and nonlinear simultaneous algebraic equations.

Such systems of equations may occur in the applied fields of physics, statistics, and industrial technology.

Solution of systems of linear algebraic equations is fundamental to the following efforts.

- Solution of vibrational problems.
- Analysis of elastic structures.
- Electrical circuit analysis and thermal analysis.
- Approximate solution of problems of elasticity.
- Approximate solution to theories of mechanics and astronomy.
- Approximate solution to problems of quantum mechanics.

Particularly for the approximate solutions above, the number of unknowns and therefore the number of simultaneous equations increases as the need for closer approximation increases. It is a rather straightforward exercise to determine how long it would take a man to solve a small system of equations. One can also determine the time and storage space needed by a computer. It is easy to see the limitation on a man in terms of attention span and susceptibility of error. The computer has, however, a somewhat different situation.

Assuming the time to completion goes as the cube of the number of unknowns, one can appreciate that time can be a limit. As more unknowns are needed to accomplish a more complete solution or a closer approximation, much more solution time is needed. All computers have a finite maximum period of time between failures. If the solution time nears or exceeds this period, additional precautions must be taken amounting to extra storage and extra time. Of course, the time taken per solution must also be reasonably compatible with the time schedule of the person requesting it!

W. J. Worlton, of the Los Alamos Scientific Laboratory, describes it as follows. "If all problems of interest to science were arranged on a scale of increasing complexity and those problems marked off that have been or can be solved with present equipment, it would be obvious that the unsolved problems are largely in the domain of higher complexity."[1] Mr. Worlton relates *complexity* to that "of the physical devices being modeled on the computer, the need for more detailed information, the increasing complexity of the mathematical models, and the growing complexity of computer hardware and software."[2] One- and two-dimensional neutronics codes and three-dimensional magneto-hydronamics codes require many more points of solution, he points out.

Also commenting on this situation, S. Fernbach of the Lawrence Radiation Laboratory says, "These problems (in mathematical physics) are for the most part describable in non-linear partial differential equations; they represent primarily the properties of materials under high pressures and temperatures as well as the transport of nuclear particles. Because of the complex geometry and multidimensional nature of these problems, the time consumed on any one run can be many hours, even on the most advanced computer. Furthermore, many runs may be necessary to optimize the parameters involved in a design study."[3]

---

[1] W. G. Worlton, "A Look Into the Future," *Nuclear News,* April 1968, page 42.
[2] *Ibid.*

Mr. Worlton further comments, "Computational physics has matured to a discipline of equal importance to theoretical and experimental physics, and the future pace of progress in research depends on using the advantages of each where appropriate."[4]

A further justification for the large computer is the relative economy of problem solution on smaller problems. There is a considerable body of evidence to support the advantage of a centralized large computer over many independent small computers. The evidence takes the form mostly of economy; that is, the large computer completes more jobs per dollar. A number of problems or jobs which do not require all the resources of the large computer may be allowed to share these resources. This *multi-programming* is a significant factor in the justification of the large computer. The question of efficiency in this sharing of resources is important to the ultimate economy. Here too, the amount of storage and the nature of the storage hierarchy, if any, plays an important part in the efficiency calculation. A major problem area is found in simply getting the problem to the computer and the results back. While the initial history of *time-sharing* of the computer has been unrewarding, and occasionally downright ridiculous, the evidence for it is too strong. Terminals connected to a large computer will unquestionably allow work to be done, which otherwise would remain untouched. Resource sharing by multi-programming is fundamental for both batch processing and on-line processing at a terminal. These terminals bring into use a number of grossly different strategies of computation and storage. Much of the time sharing controversy of the past few years reflects a lack of understanding about these new strategies. Methods of "paging" and "segmentation" are suggested. These have to do with schemes of defining and locating blocks of data. In this regard, Messrs. Harrison and Schwartz of Courant Institute indicated the following about their time-sharing system, called SHARER, implemented on the 6600, "The work described . . . leads us to question the absolute necessity of paging and segmentation hardware in a machine intended for time sharing application. Segmentation undoubtedly allows an elegant system design . . . with convenient use of reentrant coding techniques. Paging in theory should allow better use of core memory, though not to the extent that was originally hoped. Unfortunately, these advantages are often paid for in processor speed. In some cases, moreover, the potential advantages of paging seem to have been dissipated by the temptation to careless programming which a hypothetically infinite virtual memory seems to present, and by the temptation to under-design coming from architectural overoptimism."[5] This quote is given here to show that reduction to practice is a risky business indeed.

---

[3] S. Fernbach, "The Growing Role of Computers in the Nuclear Energy Field," Lawrence Radiation Laboratory, May 3, 1967, page 5.

[4] Worlton, *op. cit.,* page 43.

There is, of course, a class of problems which is essentially noncomputational but which requires a massive and sophisticated storage system. Such uses as inventory control, production control, and the general category of information retrieval would qualify. Frankly, these do not need a computer. There are, however, legitimate justifications for a large computer system as a "partner" with the computational usage.

One could argue that the economic benefit per problem of the large computer would disappear as smaller computers are improved. This would be true if smaller computers could be improved at a rate faster than the large computer. It would also be true if the large problems mentioned previously did not exist. However, neither case holds. Large computers are not at all limited in their rate of improvement, and the large untouched problems do exist. As to the first point, the author hopes that this book will show that the large computer has conceptual advantage over the small, or at least enough to encompass the small.

To sum up the justification for the large computer, the following points can be made.

- Problems are available which are substantially beyond existing computer capability.
- Some problems, due to their size, are not attempted at all.
- Processing and storage resources necessary for large problems can be used for economical solution of smaller problems.
- Large centralized computer systems can provide on-line service to terminals for a growing class of information systems, not otherwise available.

B. BUILDING BLOCKS

At the beginning of the 6600 project, the major components available and in use included:

- germanium transistors and diodes,
- air-cooled plug-in building blocks,
- ferrite magnetic cores,
- magnetic tape secondary storage.

Although in the early 1960's other components were appearing, the above were sufficiently known and understood for production and field use.

---

[5] M. C. Harrison and J. T. Schwartz, "SHARER, a Time-Sharing System for the CDC 6600," *Communications of the ACM,* X (October 1967), page 664.

Logic circuits were typically constructed in a small number of building blocks, ranging from a dozen types to three or four dozen. Most electronics for central storage units were also constructed in building block form. Varying types of design mechanization were valuable for this type of construction. Some computer-aided design was in use in which "logic equations" could be translated into wiring lists, assignment of building block types, parts totals, and so on. These schemes had four or five years of refinement and were being "fine-tuned." During this period, switching speeds of transistor circuits had rapidly improved. Since the building block approach depended on back panel wiring to accomplish the "logic," an interesting problem was appearing. With increased circuit speed, the conditions in the back panel wiring began to affect the operation significantly. This took several forms, including oscillation, noise, crosstalk, limited fan-in and fan-out at high speed, and simply the total time for transmission on the wiring.

Similarly, the central storage units using ferrite magnetic cores were experiencing significant increases in speed, as well as storage capacity. The speed increases in central storage were more difficult, however.

In spite of the substantial increases directly available in circuit and storage speed, the demands of large scale computation far outpaced any straightforward application of the faster units. It was apparent to the Control Data designers that drastic changes in approach were necessary to advance the large computer art.

C. THE APPROACH

In following chapters of this book, the approach taken in the design of the 6600 computer will be described in detail. In gross terms the approach included:

- abandoning building blocks in favor of complex, custom modules,
- moving from germanium to silicon transistors,
- moving from air cooling to freon cooling,
- adding parallel processing of functions,
- interleaving of central storage units,
- separating input-output from the central processor to ten peripheral processors,
- adding facilities for multi-programming, and
- adding magnetic disk storage to the storage hierarchy.

PACKAGING

The problems of the building block scheme required packaging more functions together to reduce back panel wiring. The result of this was a higher density of logic per unit volume. Within each more complex module the controlled electrical conditions and shorter wire lengths allowed much faster circuits. The higher density of circuits of course increases the heat density.

Fortunately, the planar silicon transistor made a timely appearance offering very high speed and higher acceptable operating temperature than the germanium equivalent.

The complex module approach to logic circuit construction also increased component density in a way which appeared to preclude air cooling. Later versions of the 6600 module, however, provided for air cooling in limited size cabinets. As will be seen in a later chapter, the conductive cooling system chosen provided a very compact high density system with very tight temperature control. The possibility of a high density compact unit offered the opportunity of substantially more logic and storage in one main frame unit.

PARALLEL FUNCTIONS—A THEORY

That there is a "theory" involved in the use of processing functions in parallel is perhaps a slight exaggeration, hopefully forgiven. A theory of multiprocessing may, for example, encompass it. What is meant here is the dimension of parallelism of function in a single job stream.

Special purpose computers utilize a fixed-wired parallelism arising from the discipline of the specialization. Certain housekeeping operations are made to operate in parallel with the main operation of the device. Many such systems perform the housekeeping functions at arbitrary points in the sequence of events. There is, in fact, a "main" sequence of operations and a "housekeeping" sequence which may be executed in parallel. There are also any number of other wired-in parallel operations of a secondary nature.

A "general" technique to accomplish the same type of parallel operation can be had with a relatively small payment in hardware. The essentials for this scheme are:

- independent functional units,
- a scratch pad,
- instruction flexibility, and
- a control system able to schedule these resources.

Functional Units can come in several varieties, including straightforward arithmetic, indexing and incrementing facility, and control of storage. Independence of operation of these units means that data may enter and leave the units rather independently and that the internal operation is independent. In order to construct a situation similar to the "main" operation and the "housekeeping" operation mentioned above, these two classes of function should be represented by separate functional units. For example, floating point arithmetic units might be applied to a "main" operation, and fixed point incremental units might be applied to the "housekeeping."

A *Scratch Pad* is a convenience for the control system and has a performance advantage over the central storage. Assignment of locations within the scratch pad can be made consistent with the usage of functional units. A sufficient number of registers, correctly applied, can provide a considerable overlap of operations, particularly reducing conflict in handling of intermediate and partial results.

Instruction Flexibility is essential in assigning registers, establishing conditions of operational overlap of units, and augmenting the natural overlap existing in an instruction stream. There are two methods for developing overlap. First is the use of complex instructions which define overlap conditions completely. Second is the use of "micro" instructions which can be arranged with flexibility to provide overlap. While the first method is completely valid, a potentially large number of instruction types are needed.

The *Control System* for the "micro" instructions is required to maintain status of the functional units, the scratch pad, and the data paths available, and to minimize the time lost in reusing these resources.

Examples of the effect of functional overlap are shown in a later chapter. In the Control Data 6600 Computer, the units, registers, and control system are designed to emphasize the flow of instructions. If an instruction calls for a unit which is not busy, the instruction is turned over to the unit whether the input data is ready or not. The premise behind this approach is that instructions following are not blocked. A later instruction, for example, may not be required to wait.

PERIPHERAL PROCESSING

While the above theory is specifically applied to the central processing unit, it can also be used in a broader context in the peripheral processing units of the 6600. These small processors can be assigned relatively independent activity either within a single job or in an environment of many jobs. In this last case, the convenience and advantage of multi-programming is effectively the overlap of jobs. As each job or task reaches a wait condition, for whatever reason, a new job can be substituted. This is a rather normal condition for a large centralized computing system servicing many users.

Attempts to separate peripheral processing from central processing include the "direct-coupled" systems or the use of a "front-end" machine. This effectively assigns the peripheral processing to one additional processor. In the 6600 the addition of ten small processors for this job opens the way to more complex and flexible system configurations.

MULTI-PROGRAMMING

A system is required, of course, for this multiple usage.

Transfer of jobs from input devices to storage, from storage to processor and back, and from storage to output devices in the presence of many other jobs is a major supervisory achievement. To prevent major obstacles and overhead in this process, a few rules are essential. Simple message formats, simple decision alternatives, and straight-forward procedures are a great help to this operation.

It is also a convenience to relieve the Central Processor Unit (CPU) from most of the burden of moving data within the storage hierarchy. For this reason, a convenient early operating system utilized a Peripheral and Control Processor (PPU) as a supervisor or monitor. In general, however, the operation of the entire system involves both the management of Input/Output (I/O) resources and the scheduling and supervision of the central resources, such as central storage and CPU. If the CPU is idle, for example, it can be used to obtain its next job, if no I/O action is required.

ORGANIZATION OF THE 6600

II

A. GENERAL

We have seen in the preceding chapter that a number of factors combine to complicate the design of the very large computer. However, the large size also gives room for design innovation. The idea of parallel functional units, for example, could hardly be tried in a small system. In spite of the complexity of the large system, there is also simplicity because there are separately defined and implemented functions. A broad description of the major elements of the Control Data 6600, shown in Figure 1, will be given in following sections of this chapter.

From this block diagram it should be clear that the connection to all external equipment is separated from both the central storage and central processor by the peripheral subsystem. Of major importance to this separation is the independent operation of the central processor and the peripheral processors. More detailed description in later sections will show the mechanics of this independent operation.

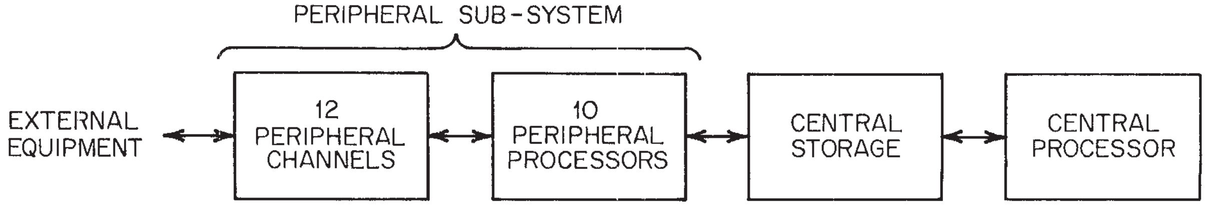

PERIPHERAL SUB-SYSTEM
EXTERNAL EQUIPMENT
12 PERIPHERAL CHANNELS
10 PERIPHERAL PROCESSORS
CENTRAL STORAGE
CENTRAL PROCESSOR

FIGURE 1

The theory behind such a separation between central processing and peripheral processing is essentially that of multiprocessing in general. Simply stated, it should be possible to accomplish a number of independent tasks with great efficiency in a set of processors connected to common storage.

For the theory to hold true, a few conditions are essential.

- There must be a number of independent tasks.
- The common storage must be able to support the data traffic.
- The time interval needed for task initiation must be much shorter than that of the task itself.

The above conditions tend to permit simultaneous processing of tasks.

In a typical usage of a large computer a single complete job may be regarded as being made up of a number of tasks. For purposes of illustration, the following list of tasks could be defined.

1. Initiate control of input device.
2. Transfer input data to input buffer.
3. Establish input file.
4. Perform n computational tasks.
5. Establish output file.
6. Initiate control of output device.
7. Transfer output data to output buffer.
8. Transfer output data to output device.

From this example, the idea of assigning a number of small processors to the peripheral tasks would appear valuable, particularly if there is a flow of jobs available to the system.

B. PERIPHERAL SUBSYSTEM

Ten small processors are included in the 6600 Computer, as shown in Figure 2. They are called Peripheral and Control Processors (PPU).

Each of these ten small processors contains a private storage unit and an arithmetic and control capability. Each processor has access to the Central Storage and to the peripheral channels. Some of these properties are listed below. Each PPU has:

- Storage unit of 4096 12-bit words.
- Storage unit cycle time of one microsecond.
- An accumulator register of 18-bit length.
- A repertoire of sixty-two instructions.
- Ability to transfer one word or a block of words to or from central storage.
- Ability to transfer one word or a block of words to or from a peripheral channel.

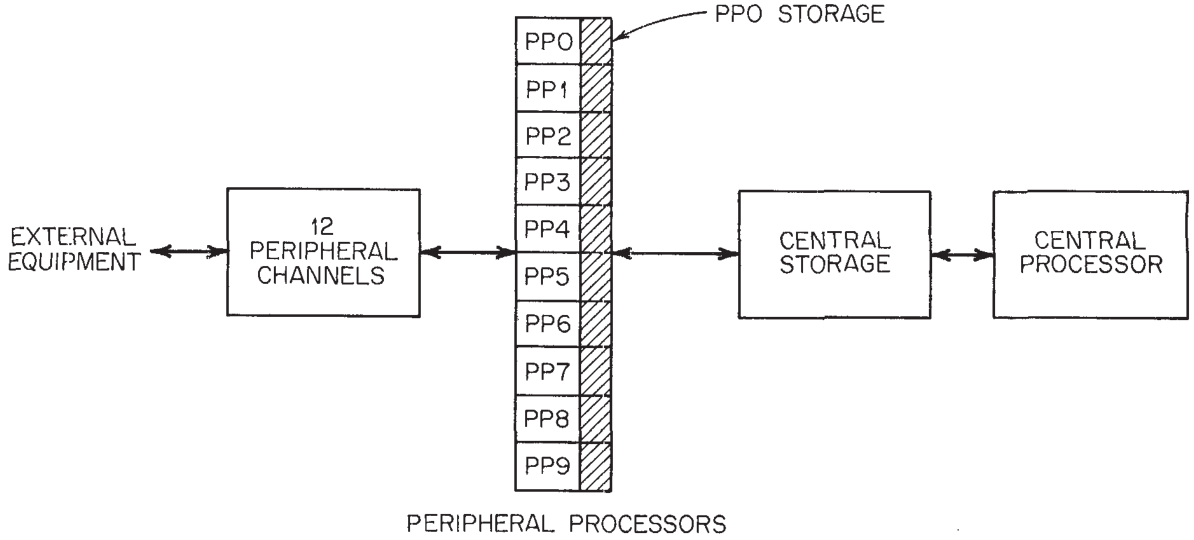


FIGURE 2

Each PPU executes an independent stored program located in its private storage unit. These stored programs are loaded by an operator by means of an operation called DEAD START. Details of this operation are described in a later chapter.

Each PPU can communicate with any of the other nine in two ways, central storage or a peripheral channel. For this communication to occur, each processor involved must "cooperate" by means of its stored program.

Each PPU can communicate with the Central Processor Unit (CPU) in two ways, central storage and "exchange jump." For communication to occur through the medium of Central Storage each processor involved must "cooperate" by means of its stored program. In this case, one of the stored programs is in the PPU, and the other stored program for the CPU is located in Central Storage. The "exchange jump" signal is "one way" in that any PPU may cause the CPU to halt its current program and begin a new one. In this case, "cooperation" by the CPU is unnecessary since the exchange jump is a hardware property.

The Peripheral Processors may independently utilize any of the Peripheral Channels. For illustration a printer is shown connected to a Peripheral Channel and, in turn, to a PPU in Figure 3 (page 12).

It should be noted that no fixed relationship exists between Peripheral Channels and Peripheral Processors. In this illustration, Figure 3, channel 4 is connected to the printer and PPU 2 is controlling the device. On completion of the printing operation, PPU 2 may be reassigned to another task and to another channel. Similarly, several devices may be connected to a channel. On completion of the printing operation, channel 4 may be assigned to another device connected to the channel.

In this case a printer, as shown, can be driven from either of two channels, channel 4 or channel 8. The advantage of this technique is dependent on the kind of system configuration desired.

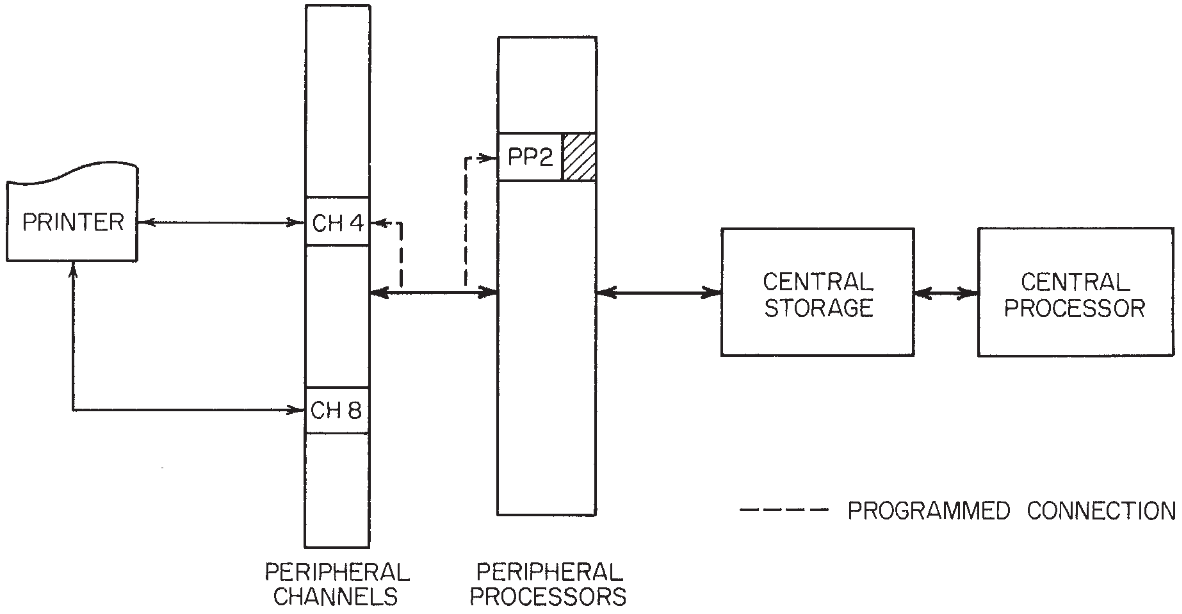

PRINTER
CH 4
CH 8
PP2
CENTRAL STORAGE
CENTRAL PROCESSOR
PROGRAMMED CONNECTION
PERIPHERAL CHANNELS
PERIPHERAL PROCESSORS

FIGURE 3

C. CENTRAL PROCESSOR—CPU

The Central Processor of the Control Data 6600 Computer is based on a high degree of functional parallelism. This is provided by the use of many functional units and a number of essential supporting properties, as shown in Figure 4.

The ten functional units are independent of each other and may operate

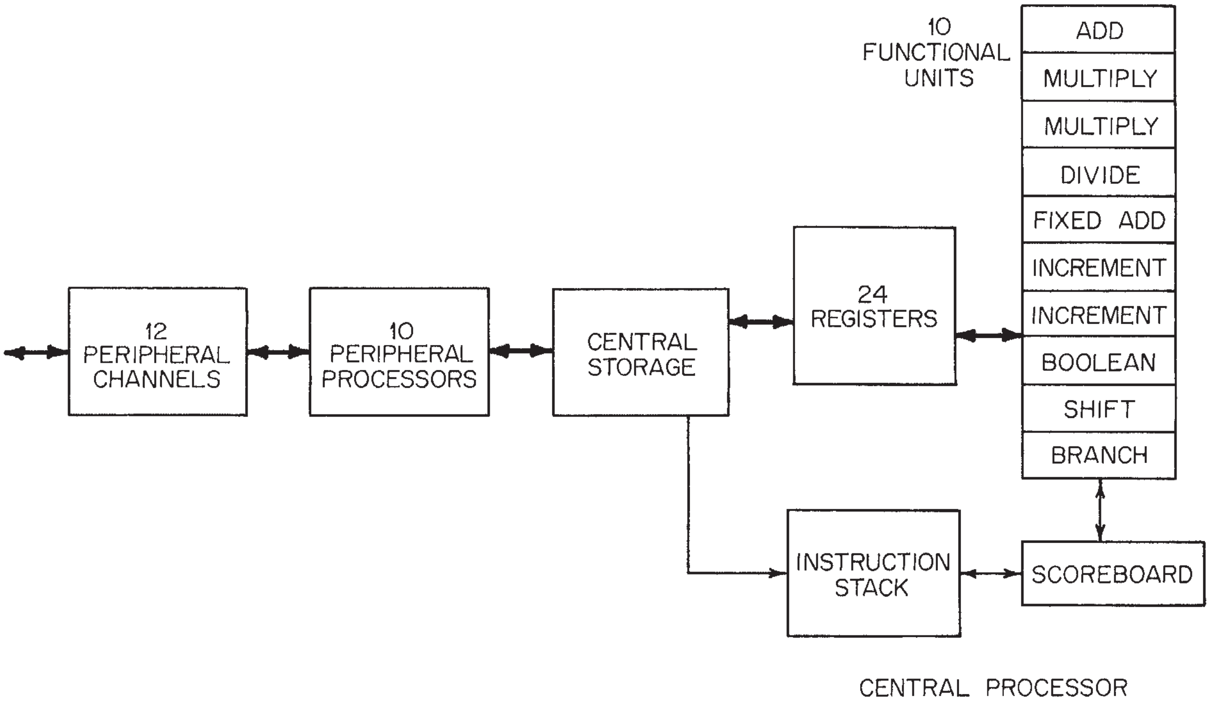

10 FUNCTIONAL UNITS
ADD
MULTIPLY
MULTIPLY
DIVIDE
FIXED ADD
INCREMENT
INCREMENT
BOOLEAN
SHIFT
BRANCH
12 PERIPHERAL CHANNELS
10 PERIPHERAL PROCESSORS
CENTRAL STORAGE
24 REGISTERS
INSTRUCTION STACK
SCOREBOARD
CENTRAL PROCESSOR

FIGURE 4

simultaneously. In a typical central processor program at least two or three functional units will be in operation simultaneously. The ten units are:

- Floating Add
- Floating Multiply (2)
- Floating Divide
- Fixed Add
- Increment (2)
- Boolean
- Shift
- Branch

Twenty-four registers are included in the Central Processor. Eight of these are assigned as operands or data words and are sixty bits in length. Eight are assigned as index registers and are eighteen bits in length. Eight are assigned as address registers and are eighteen bits in length. All arithmetic functions are executed on operands from the registers with results returned to the registers. The selection of the sixty-bit length was made for efficient instruction packing and for extended floating point precision. The eighteen-bit registers provide a convenient size for address manipulation.

Instructions in the CPU are three address in general, one register address for each of two operands and one result. For example, the equation

$$A = B + C$$

contains two operands, B and C, a function +, and a result A.

The use of registers in the Central Processor allows for convenient handling of partial or intermediate results. Central Storage could, of course, be used for these values. However, a store operation followed by a fetch operation would be required with a significant time penalty.

Instructions are loaded into the CPU in sequence from Central Storage under control of a Program Address Register. As the CPU program proceeds, up to a maximum of seven "old" instruction words are saved. Under some circumstances, these old instructions can be reused without referencing memory. An obvious case is shown below.

Location	Contents
Program Address n	Instruction Word n
Program Address n + 1	Instruction Word n + 1
Program Address n + 2	Instruction Word n + 2
Program Address n + 3	Instruction Word n + 3
Program Address n + 4	Conditional Branch to n

In this example a conditional branch instruction in program address location n + 4 calls for a "loop" back to location n. Under the correct circumstances, this entire loop can easily be held within the instruction stack. The program can loop within the stack itself at high speed without requiring any storage references for instructions. There are two advantages to this case. First, the instruction fetch is much faster. Second, fewer storage conflict conditions are possible since fewer actual storage references are made.

Instructions are introduced to the control system in sequence. A simple test is made in a unit called the SCOREBOARD, after which the instruction is issued to the appropriate functional unit or is held until the test can be passed. The test determines if the functional unit is busy and if the register assigned for the result is not reserved. Instructions may be issued at a very high rate, held back only by the unit busy or register reserved condition. With a number of functional units and a number of registers available, the probability of high issue rates is reasonably good even without any optimization efforts.

Data transfer occurs between the Central Storage and Central Processor on a number of separately controlled paths. Five of the 60-bit registers are assigned as read registers and two as "store" registers. This reflects a typical unbalance of traffic between read and store. Address registers are assigned one-for-one with each of these read and store registers. In order for a storage reference to be initiated for a data transfer, the specified address register is set to the desired address by a CPU instruction. This new address is used to reference storage for a read or a store depending on which address register was set. The data will enter or leave the operand register in a "partner" relationship with the address register, as shown in Figure 5.

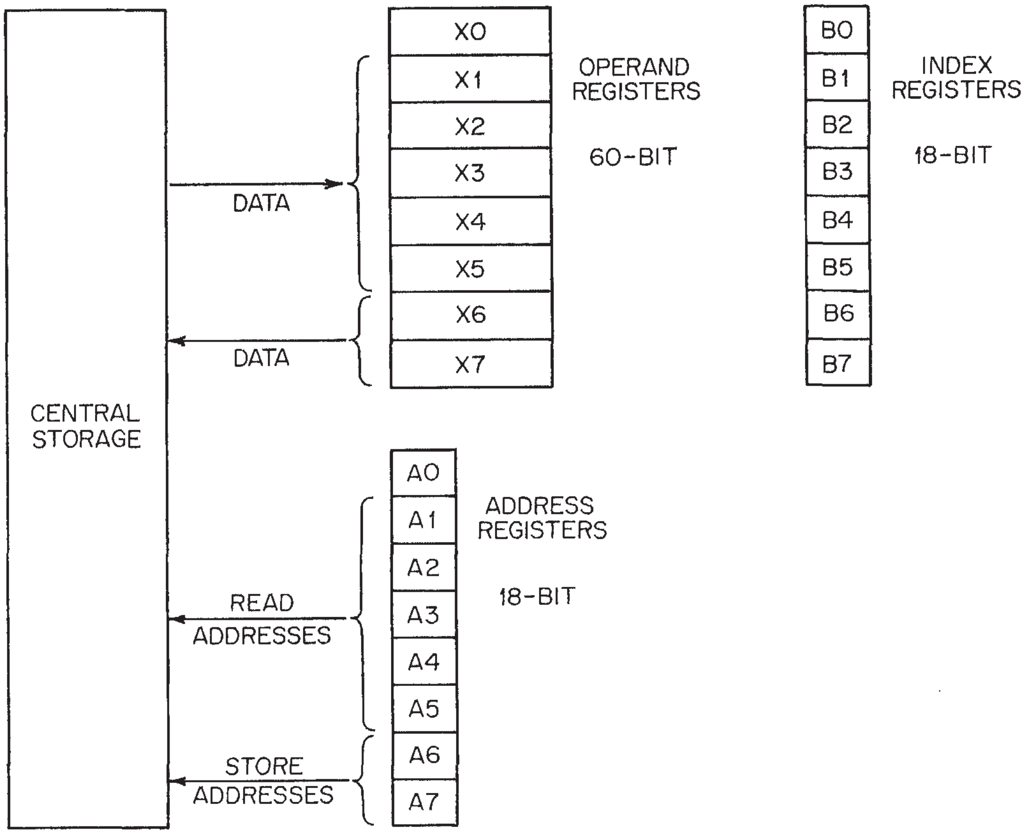

X0
X1
X2
X3
X4
X5
X6
X7
OPERAND
REGISTERS
60-BIT
B0
B1
B2
B3
B4
B5
B6
B7
INDEX
REGISTERS
18-BIT
DATA
DATA
CENTRAL
STORAGE
A0
A1
A2
A3
A4
A5
A6
A7
ADDRESS
REGISTERS
18-BIT
READ
ADDRESSES
STORE
ADDRESSES

FIGURE 5

In summary, the CPU contains five essential ingredients for parallel execution of a single stream of instructions. These are:

- Ten independent functional units,
- Twenty-four registers,
- A control system with scoreboard,
- An instruction stack,
- Multiple paths to Central Storage.

D. CENTRAL STORAGE

Any large-scale computer is critically dependent on a powerful central storage system. In spite of methods which tend to reduce the number of references to the Central Storage, the remaining references have a dominating effect on the processing speed and system throughput. The organization used in the 6600 Central Storage is the result of a sensitive balance of physical and economic considerations to serve the requirements of the CPU and the peripheral subsystem.

Design of a high-speed storage unit is affected by the following considerations which may be termed axioms.

1. Storage cycle time tends to increase directly with the size of the storage unit.
2. Storage cost per bit tends to increase inversely with the size of the storage unit.

The design is forced into a storage hierarchy by these conditions, especially as the need for more storage grows. The 6600 Central Storage was limited to 131,072 words with a second level in the storage hierarchy supplied by Extended Core Storage. A substantial economic differential exists between these two levels.

Shown in Figure 6 is the system interconnection of the Central Storage. Control of storage references to Central Storage is provided by the STUNT BOX.

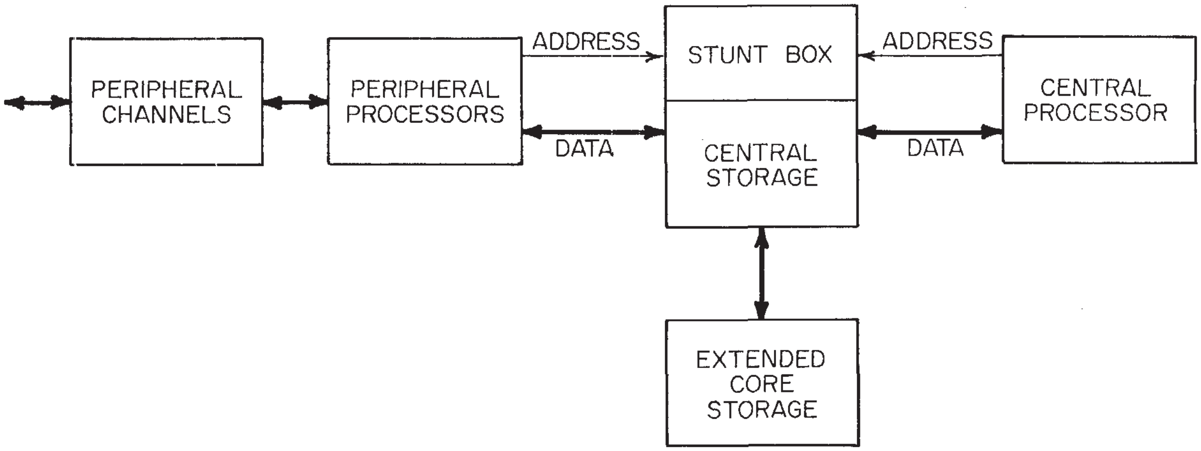


FIGURE 6

The Central Storage of 131,072 60-bit words is constructed in 32 independent banks. These banks are arranged in an interleaved fashion which provides a high degree of random access overlap and block transfer speed.

An address of seventeen bits is split as shown in Figure 7. The least significant five bits of the address are used to define the bank. If the address is repetitively increased by one, all 32 banks will be referenced before returning to the first bank. For block transfers, this allows the storage cycle to be 32 times longer than the time required to transfer a single data word. As a practical matter, other factors tend to establish the relationship of the storage cycle with transfer cycles.

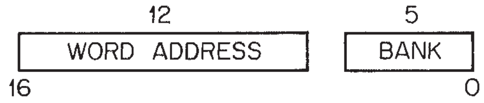


FIGURE 7

Two cycles are defined in the computer. The first of these, the MAJOR CYCLE, is identical with the storage cycle of the PPU storage unit and the Central Storage unit. The second, the MINOR CYCLE, is a measure of the time taken to transfer one data word through the storage distribution system. As will be seen in later chapters, most operations are directly related to the MINOR CYCLE.

MAJOR CYCLE—1000 nanoseconds, or one microsecond.
MINOR CYCLE—100 nanoseconds.

The STUNT BOX is designed to provide a maximum flow of addresses to the Central Storage. Occasions in which an address is being held because of a bank-busy condition do not stop other addresses from passing. Implementation of this unit in the 6600 is especially dependent on the synchronous and predictable nature of the Central Storage system.

The storage unit making up both the Central Storage system and the Peripheral Processor "private" storage is a magnetic core, coincident-current unit. As will be shown, this unit is modular and pluggable, with the following properties.

- Storage Read and Store Cycle—1000 nanoseconds,
- Word Length—12 bits,
- Capacity—4096 words,

Central Storage banks require five such units making up a word length of 60 bits and a capacity of 4096 words per bank.

The protection of data or programs held in the Central Storage is accomplished by the Central Processor and Peripheral Processors independently. The Peripheral Processors are instruments of the operating system and, as such, have access to Central Storage only by assignment. The Central Processor is an instrument of the operating system at one time and under control of a "user" program at other times. In either situation, the Central Processor is allowed access to an area specified by the operating system.

E. EXTENDED CORE STORAGE

A new element of computer storage hierarchy is the Extended Core Storage (ECS). It would be an understatement to point out that this element is an unknown factor in the performance or economy of a computing system. This unit was added to the 6600 computing system, well after first deliveries, in an effort to smooth the storage hierarchy. Studies of actual practice should place a proper perspective on this unit. In any case, early usage coupled with simulation studies show it to be an important unit indeed.

The next level of storage hierarchy has been a rotating magnetic device, such as the magnetic drum or disk. While these are valuable, there is considerable performance difference between them and the Central Storage. This "gap" is, and will be, a target for inventive offerings. A later chapter will detail the nature of the various devices contributing to this gap.

A primary goal of the Extended Core Storage is simply the economic enlargement of Central Storage. While direct random access of the extended storage is a valid and acceptable use, a particular advantage exists in block transfers between Central Storage and Extended Core Storage. This advantage is a by-product of the specific properties of the Extended Core unit. Some of the more pertinent characteristics are:

- Extended Read and Store Cycle—3.2 microseconds,
- Storage Word Length—480 bits,
- Bank Capacity—125,000 "central" words (60-bit),
- Number of Banks—Up to 16,
- Interface Trunk Width—60 bits,
- Interface Trunk rate—10 "central" words per microsecond.

As will be shown in detail in later chapters, this transfer rate is capable of matching the maximum transfer rate of the Central Storage. As a result, block transfers between these two storages can proceed at maximum system rate. This transfer rate is some fifty times the equivalent rate through a Peripheral Channel.

The diagram in Figure 8 shows the logical organization of the Extended Core Storage connected to two 6600 Computers. The Central Processor initiates any transfer, whether it is a single 60-bit word or a block transfer. Control is given to a unit called the ECS Coupler, which establishes control over both the Central Storage and the Extended Core Storage for the complete transfer.

Shown in the diagram are four sets of one-half million words, each under control of a unit called the ECS Controller. This unit allows connection of four access channels similar to the ECS Coupler connection. It can be

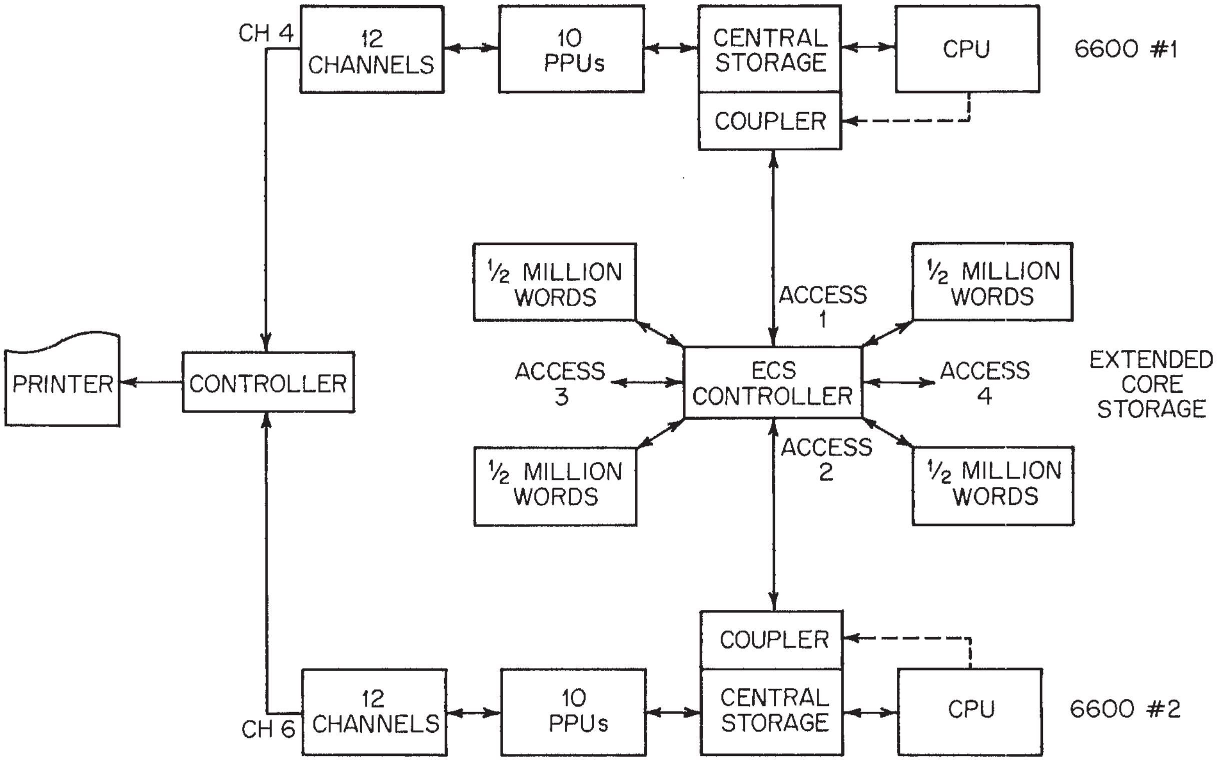

CH 4
12 CHANNELS
10 PPUs
CENTRAL STORAGE
CPU
6600 #1
COUPLER
½ MILLION WORDS
ACCESS 1
½ MILLION WORDS
PRINTER
CONTROLLER
ACCESS 3
ECS CONTROLLER
ACCESS 4
EXTENDED CORE STORAGE
½ MILLION WORDS
ACCESS 2
½ MILLION WORDS
COUPLER
12 CHANNELS
CH 6
10 PPUs
CENTRAL STORAGE
CPU
6600 #2

FIGURE 8

seen that this type of organization allows the Extended Core Storage to be a central "common" storage for a system of multiple processors.

Note that each computer utilizes an ECS Coupler to control its transfers, with the ECS Controller handling the storage unit. For illustration purposes, a printer is also shown with a "dual channel" controller connected to a Peripheral Channel from each computer.

Storage protection in Extended Core Storage is accomplished in a manner similar to Central Storage. Since the Central Processor of each attached 6600 Computer is the only unit making reference, the protection mechanism can be located in the CPU. This mechanism is separate from that providing protection for the Central Storage.

The CPU initiates transfers by defining an initial address in Central Storage and an initial address in Extended Core Storage. The length of block is specified in the CPU instruction; whereas the initial addresses are defined by address register A0 of the CPU and its "partner" operand register. Note that ECS requires an address register of 21 bits in order to define two million locations. Similar to the Central Storage, a group of storage banks is interleaved for block transfer performance advantages. This interleaving is limited to four banks, however.

A principal usage of Extended Core Storage involves "swapping" of programs or data between the Central Storage and ECS. A theoretical advantage can be claimed for holding segments of programs in the central storage because of the time penalty, or overhead, of swapping. Therefore, the very high transfer rate of ECS has a particular advantage, whether swapping is a primary strategy or not.

BASIC CIRCUIT PROPERTIES

III

Although it is not necessary to know the intimate engineering details to study the logic of computers, the knowledge gives additional insight into the underlying reasons for the design. For it should be remembered that even the most exquisite piece of logic must be fitted into the physical ground rules in order to be put to work.

A. THE SILICON TRANSISTOR

There is a striking chronological relationship between the appearance of the silicon planar transistor and the Control Data 6600. This component was much sought after for many early years of transistor development. However, the methods used in those early years did not succeed with silicon. As a result, early transistor computers used germanium which is perfectly satisfactory under controlled environment. Upper limits of temperature during manufacture and during operation are very much lower for germanium than silicon.

When the planar process was invented, the advantages of silicon were open to use. Of particular importance to computer circuits are the higher junction temperatures allowable and generally higher current and power levels allowable.

A second important value resulting from the planar process is the higher device speed. This can be attributed to the use of the NPN configuration for the transistor, a configuration that had been difficult to obtain. While the PNP configuration had been useful, it was inferior in charge storage, thus limited in high-speed switch usage. [Note: The letters NPN refer to impurity characteristics of the collector, base and emitter respectively. "N" specifies an excess of electrons, thereby implying a negative charge. "P" specifies a lack of electrons, thereby implying a positive charge.]

In terms of economics, reliability, and device performance, the silicon transistor has been a success. The manufacturing yield of the transistor is a rather sensitive function of surface area of the silicon used. At the very small sizes problems of mask alignment, handling, and packaging combine to preclude the use of dice smaller than 10 to 20 mils on a side. [Note: Dice refers to the final "chip" of silicon into which are diffused the transistor elements.]

Figure 9 shows the yield as a function of utilized area. As the area increases past the peak yield, the constants k_1 and k_2 for the exponential portion of the curve depend on the average number of defects per unit area and the process used to manufacture the device. As a practical matter the peak attainable yield is a reasonably good fit with a high-speed transistor.

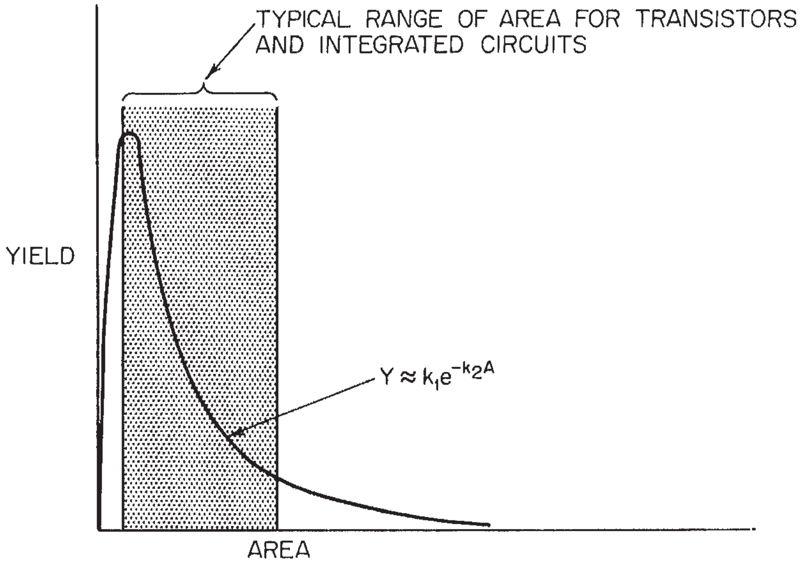


FIGURE 9 Relationship between yield and circuit area in integrated circuit production.

Figure 10 shows the improvement in reliability of transistors during the ten-year interval 1954 to 1964. According to this curve, the failure rate is approaching a limit value of 4×10^{-9} failures per hour, or a more familiar figure of 0.0004 percent per 1000 hours.

This rate of failure is lower than that of the interconnections between components in a computer circuit. Since the entire 6600 Computer contains approximately 400,000 transistors, the system mean free time between failure due to the transistor is over 2000 hours.

The silicon transistor used is made beginning with an intrinsic-type wafer of silicon. An epitaxial layer of n material is grown on the surface of

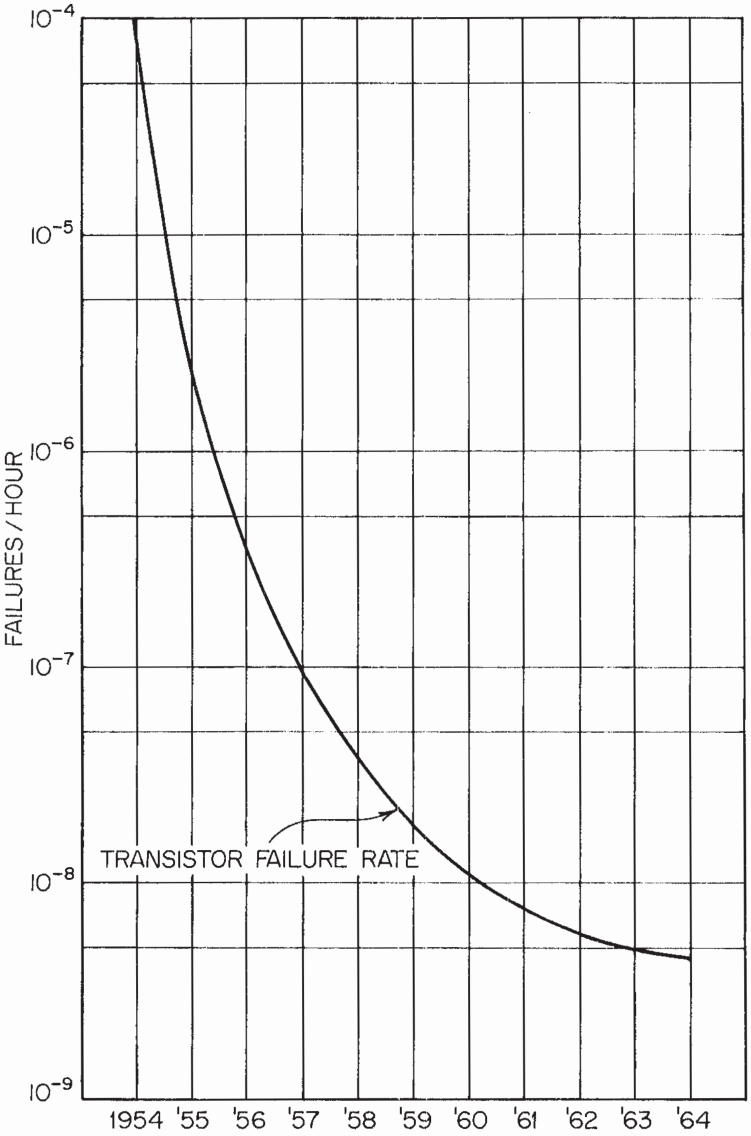


FIGURE 10 Reliability improvement in transistors over the ten-year period 1954–1964.

the wafer, and the elements of the transistor are made by using a photographic masking technique to permit successive diffusions of alternating p and n material into the epitaxial layer. EPITAXY is a method for obtaining a constant impurity concentration in a very narrow layer and, in the case of this transistor, is important to its speed characteristics. (See Figure 11, page 22.)

B. DCTL LOGIC CIRCUITS

The basic logic circuit used in the 6600 Computer is the Direct-Coupled Transistor Logic circuit, abbreviated DCTL. This is one of the simplest switching circuits devised and is heavily dependent on the transistor characteristics for its operation. The basic inverter is shown in Figure 12 (page 22).

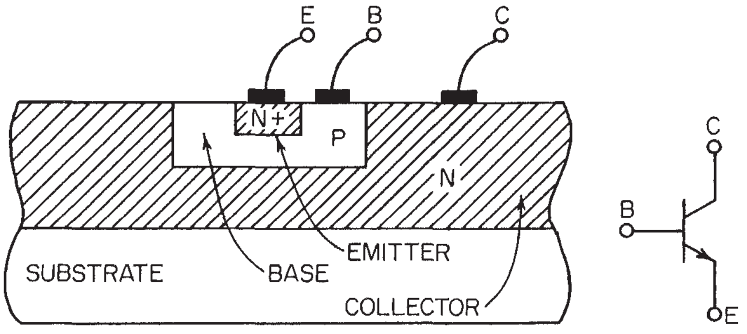


FIGURE 11

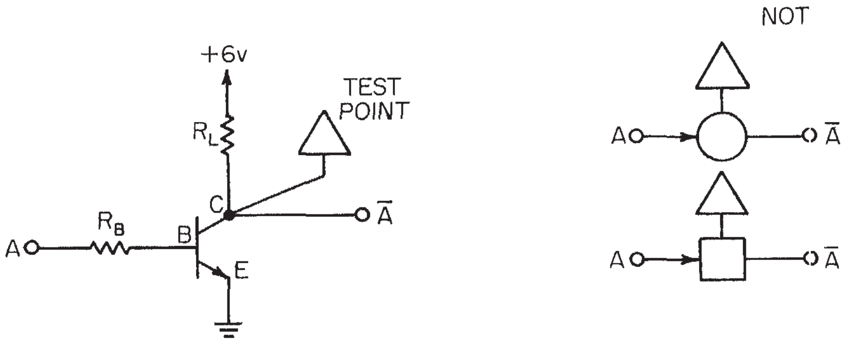


FIGURE 12

The circuit provides a logical inversion (NOT), and the output may be used to drive several similar circuits. Signal levels, as seen at either an input or an output point, are either +0.2 volt or +1.2 volt. Two symbolic representations for this circuit are also shown in the figure. A +1.2 volt signal at the input turns on the transistor and drives it into saturation. The condition of saturation is such that no amount of additional base-emitter current will cause the collector-emitter voltage drop to go any lower. This is a limit condition and results in charge storage in the transistor. The approximate value of the collector-emitter voltage drop is +0.2 volt.

With a +0.2 volt input, the collector voltage rises toward +6 volts, but is limited to +1.2 volt by current flowing in the base of load transistors being driven by the circuit.

The threshold voltage at the base of the transistor is approximately +0.7 to +0.8 volt. Below this value virtually no current flows in the transistor. Above this value, current flows in the base-emitter path and in the collector-emitter path. Circuit parameters for the cutoff and saturation conditions are listed below.

	V_B	V_C	I_B	I_C	Average Switching Time
Cutoff	+0.2	+1.2	0	0	
Saturation	+0.8	+0.2	1 ma	10 ma	5 nanoseconds

The selection of the logical representation of the circuit allows for two possibilities. For example, in the circuit of Figure 13, two truth tables may be made.

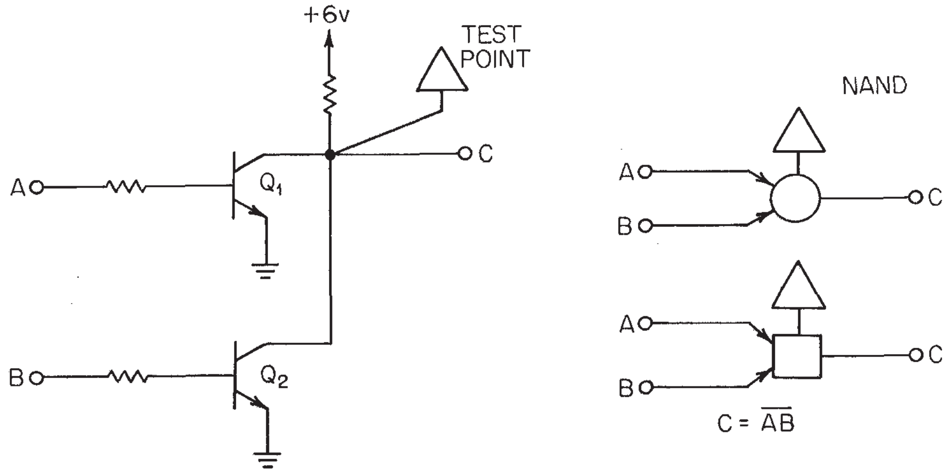


FIGURE 13

The two tables shown below represent an arbitrary definition of the signal levels to the logical values "1" and "0." Table I defines input logical "1" as +0.2 volt and output logical "1" as +1.2 volt. Table II defines input logical "1" as +1.2 volt and output logical "1" as +0.2 volt. In other words, the logical value inverts between input and output.

	Actual Conditions				Table I				Table II			
A	0.2	1.2	0.2	1.2	1	0	1	0	0	1	0	1
B	0.2	0.2	1.2	1.2	1	1	0	0	0	0	1	1
C	1.2	0.2	0.2	0.2	1	0	0	0	0	1	1	1

Assume that the two logical representations of Figure 13 are assigned to the two tables, the "circle" to Table I and the "square" to Table II. It should be clear that the result of the circle is C = AB and that the result of the square is C = A + B.

It should be an interesting exercise for the student of Boolean algebra to prove this without recourse to the artifice of "inverted" definitions. Before getting away from this, however, the reader should observe that measurements taken at test points will tend to follow nicely this mental gyration.

Since a computer design is made up of fairly simple combinations of AND, OR, and NOT, it is instructive to show two such combinations in Figures 14 and 15. These two cases utilize the definition of logical value "1" as +0.2 volt for the points labeled A, B, and C.

Note that this remains consistent with the previous artificial definition. These two figures illustrate another characteristic of the DCTL circuit. Brief mention was made about driving several "load" circuits. This facility

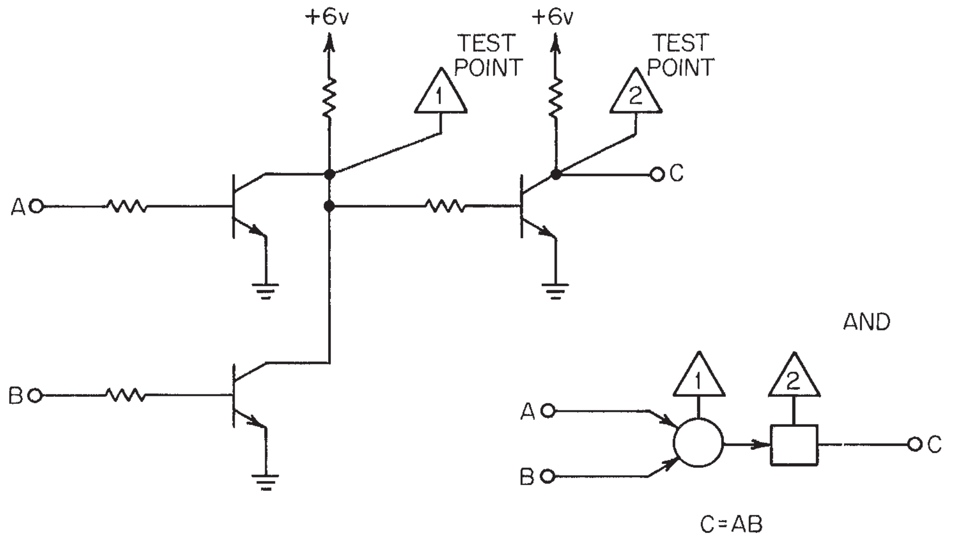

+6v
TEST POINT
1
+6v
TEST POINT
2
C
A
B
AND
1
2
A
B
C
C=AB

FIGURE 14

is equally available in either the "circle" or the "square" configuration. It should be clear then that the value $\overline{AB}$ is available in Figure 14 and the values $\overline{A}$ and $\overline{B}$ are available in Figure 15.

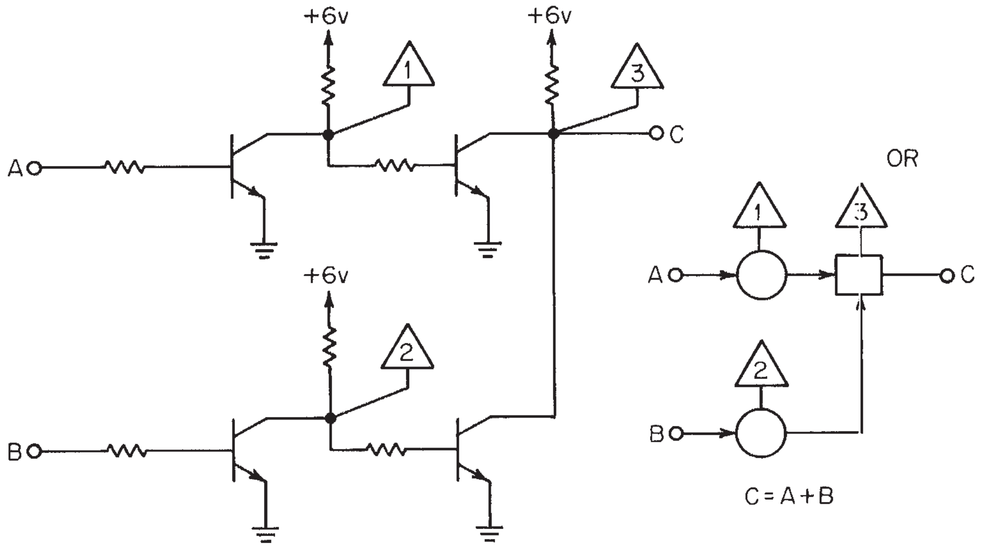

+6v
1
+6v
3
C
A
+6v
2
B
OR
1
3
A
C
2
B
C=A+B

FIGURE 15

C. LOGIC SYMBOLS

The logical representations used in the preceding figures are based on a very simple relationship with the electrical components used.

Symbol	Logical Function	Electrical Component
→	Inversion	Transistor (Including Its Base Resistor)
○	Usually AND Combination	Collector Load Resistor
□	Usually OR Combination	Collector Load Resistor

The apparent advantage to the designer of knowing from the logic diagram an exact count of electrical components is specific to this type of logic.

Figure 16 shows the representation of the exclusive OR function using the symbols described above. In later chapters it will be seen that the "intermediate" values available in the combination of Figure 16, specifically the outputs of the "circles," can be powerful uses of the DCTL circuits.

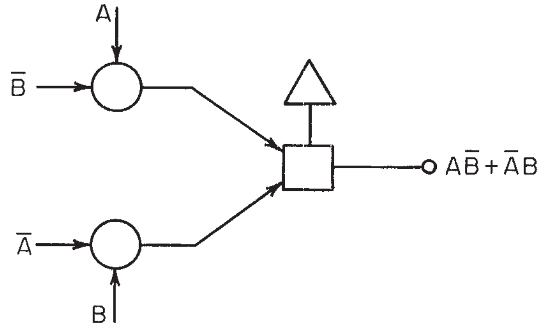


FIGURE 16

At this point, it is convenient to describe some of the ground rules of use of the DCTL circuits, as defined in the 6600 Computer. One circuit which may be a common collector connection of several transistors can drive several other transistors, either on the same module or on a separate module.

A circuit may drive up to five transistors when all transistors, including driver, are physically on the same module. This loading limitation is found experimentally from variations in base threshold voltage and corresponding base current demand in the load transistors. It is also affected by the impedance level of the etched copper printed circuit interconnection within the module. For each load configuration the driver collector resistor R_L and base resistors R_b of the driven transistors are adjusted to limit and balance the base currents.

Other load limits apply when the circuit load is on a separate module, requiring back panel wiring. A circuit can simultaneously drive up to two transistors on a separate module. Loading in this case accounts for the loss on the transmission line connecting the two modules as well as variations in base threshold voltage.

An additional limit of six collectors connected together for OR and AND functions completes the very simple list of constraints. This limit is required for speed reasons only and represents the maximum capacitance acceptable at a collector point. The time constant "t" may be calculated for this point from the capacitance per collector and the collector load resistor R_L. Using the highest value of R_L of 680 ohms and 2 micro-micro-farads for each collector, the time constant for six collectors is 8.16 nanoseconds.

The exponential charging of the collectors toward the power buss (+6 volt) is held at the value +1.2 volt by load currents. Therefore, a small

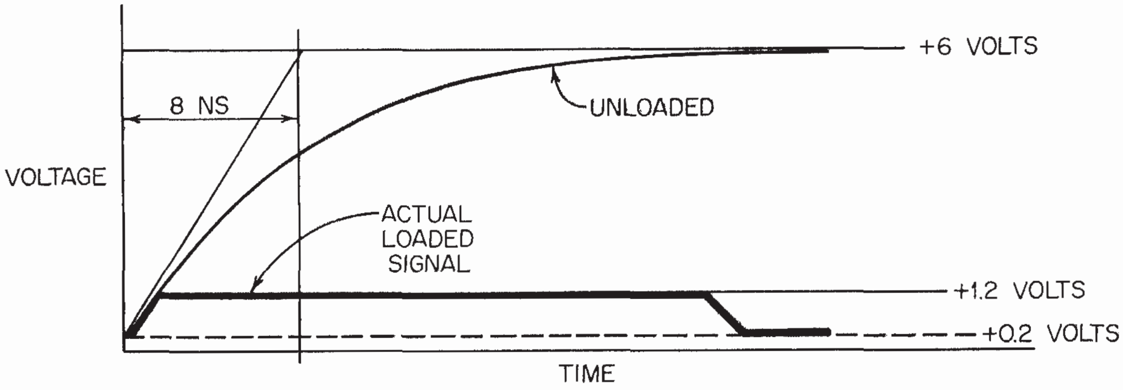


FIGURE 17

fraction of the 8.16 nanoseconds is consumed for the "turn-off" case, as shown in Figure 17. In fact, the rise time due to capacitance is shorter than the transistor turn-off characteristic. One can easily see that a faster transistor would require changes in this rather comfortable balance in order to be effective.

The circuit of a flip-flop and its clear/set input are shown in Figure 18. This basic flip-flop may have up to five set inputs and five clear inputs; in this example only one of each is shown. The set input is fed from a three-way AND gate, of which one input comes from the clear/set network.

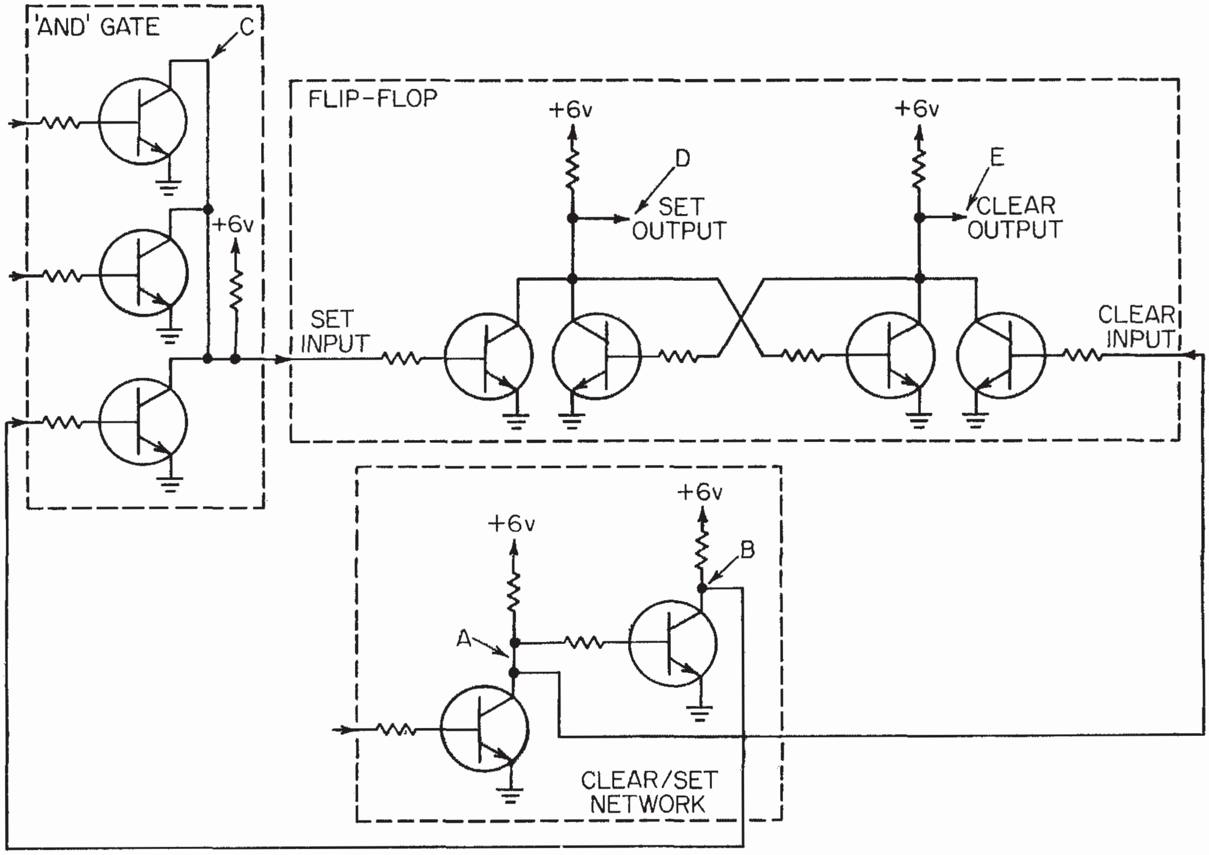

'AND' GATE
C
+6v
FLIP-FLOP
+6v
+6v
D
SET
OUTPUT
E
CLEAR
OUTPUT
SET
INPUT
CLEAR
INPUT
+6v
+6v
B
A
CLEAR/SET
NETWORK

FIGURE 18 Basic flip-flop with clear/set input.

The clear/set network enables the flip-flop to be cleared and reset by the same gating pulse. In most cases this will be a twenty-five nanosecond clock pulse. Figure 19 shows a logical representation of this circuit and a timing diagram for the clear/set operation.

The previous definitions of logical "1" are required. All input and output connections as shown in Figure 19 define a "1" as +0.2 volt.

A useful symbolic treatment is shown in this figure with the letters "A" through "E" labeling the circles and squares, as shown. For clarity these are shown also in Figure 18. The timing chart assumes an inverter delay of five nanoseconds in all cases.

One important characteristic of this circuit is that the set output is

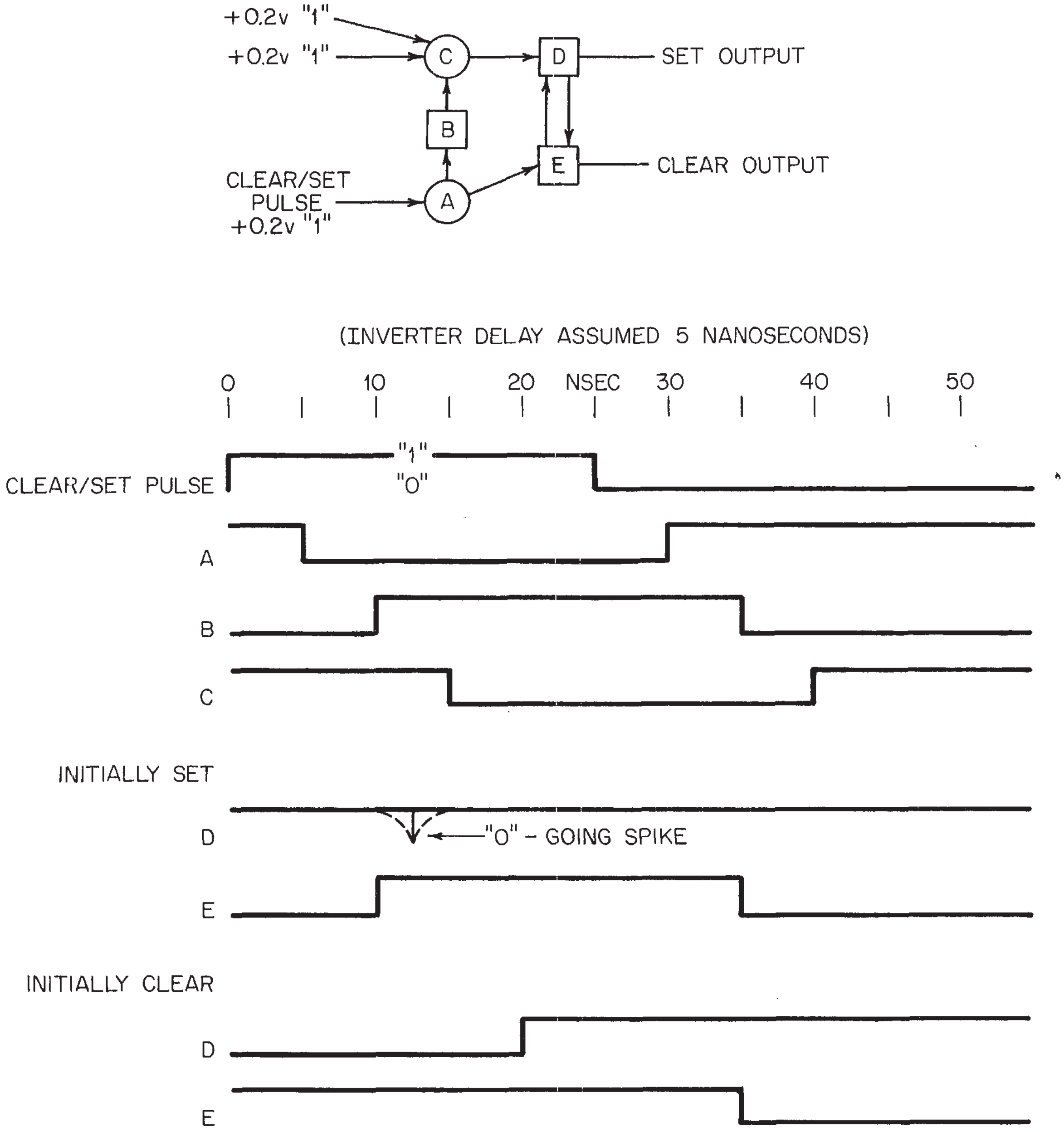

+0.2v "1"
+0.2v "1"
C
D
SET OUTPUT
B
E
CLEAR OUTPUT
CLEAR/SET PULSE +0.2v "1"
A
(INVERTER DELAY ASSUMED 5 NANOSECONDS)
0
10
20
NSEC
30
40
50
CLEAR/SET PULSE
"1"
"0"
A
B
C
INITIALLY SET
D
"0" - GOING SPIKE
E
INITIALLY CLEAR
D
E

FIGURE 19 Flip-flop logical representation and timing chart.

available immediately after the inverter delays represented by "C" and "D." However, the clear output is not available until after the inverter delays following the trailing edge of the clear/set input signal.

The spike shown for the case of the flip-flop initially set is nominally five nanoseconds wide, representing the difference in the path A-E-D and A-B-C-D.

A summary of design constraints is:

- a collector can drive five bases in a module,
- a collector can drive two local bases in a module and two bases by back-panel twisted pair on one other module,
- six collectors can be connected within a module.

D. TRANSMISSION LINES

For back-panel interconnections and for chassis-to-chassis interconnections, two kinds of circuits are used. At the high frequencies used, these interconnections must be treated as transmission lines and protected from anomalous behavior.

Within one chassis it is convenient to interconnect using a "DC" form of circuit since the DCTL logic within the module can then be easily treated between modules. Distances of the wiring within a module are two to three inches maximum with transmission velocities of about 0.1 nanosecond per inch. Distances external to the module but within a single chassis can range up to about five feet, with transmission time of about 1.3 nanosecond per foot. Since the circuit speed is in the range from three to five nanoseconds, this distance must obviously affect the design.

The twisted pair driver is shown in Figure 20. Initially, consider Q1 conducting so that its collector is at +0.2 volt. With Q2 turned off, the

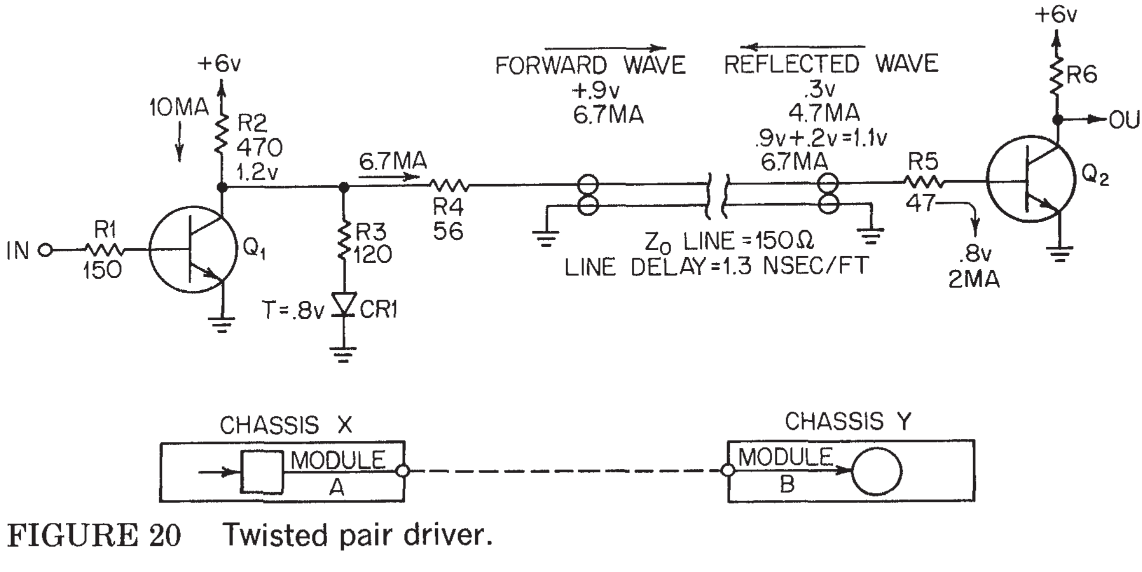

+6v
10MA
R2
470
1.2v
6.7MA
R4
56
FORWARD WAVE
+.9v
6.7MA
REFLECTED WAVE
.3v
4.7MA
.9v+.2v=1.1v
6.7MA
R5
47
.8v
2MA
+6v
R6
OUT
Q2
IN
R1
150
Q1
R3
120
T=.8v
CR1
Z0 LINE =150Ω
LINE DELAY=1.3 NSEC/FT
CHASSIS X
MODULE
A
CHASSIS Y
MODULE
B

FIGURE 20 Twisted pair driver.

transmission is terminated as an open circuit. Since this line will not go anywhere else, the open termination is allowed, and a termination for the reflected signal is needed at the sending end.

A +0.2 volt input causes the collector of Q1 to rise toward +6 volts. Current splits between the resistor-diode leg and the twisted pair line, with 6.7 milliamperes out of the total of 10 ma being sent into the line. This value of current is found from the resistance values, the forward characteristics of the diode, and the surge impedance of the transmission line. This last value is in the range of 130 ohms to 150 ohms.

A wavefront of voltage and current is sent down the twisted-pair transmission line, with the incremental voltage adding to the rest state of the line of +0.2 volt. At the base of Q2, the voltage of this wavefront exceeds the threshold of the base, causing base-emitter current to flow and turning on Q2. This current does not represent a perfect termination for the line; therefore, a reflected wave of voltage and current is sent back through the twisted-pair transmission line. When this reflected wave front reaches the sending end, the resistor-diode network appears as a perfect termination. Therefore, no further reflections are introduced, and the line is stable at +1.1 volt.

In the reverse case in which Q1 is turned on, a similar wave is sent down the line and is, in turn, reflected. The reflected wave returns to the sending end to find the least resistance path to be the collector-emitter path of Q1. The resistor in series with the line then serves to correct the network to a perfect termination.

The shunt diode and resistor are replaced by a transistor and its base resistor when it is necessary to drive a circuit on the same module. The circuit operates in the same manner since the base-emitter circuit of the transistor acts as a diode.

The symbols used for the twisted pair driver and receiver are squares and circles as in the rest of the logic, with a small circle signifying the module connection.

Interconnection between major physical entities, such as chassis or cabinets, is accomplished by pulsed transmission over coaxial cables. Principal reasons for this are related to self-induced electrical noise and protection against external electro-magnetic noise. The coaxial transmission circuit is shown in Figure 21.

Initially, Q1 and Q2 are turned off so that no current flows in the primary winding of the transformer. When Q1 is turned on, a current flows from the +6 volt supply through one half of the primary winding; a current also flows in the secondary winding by transformer action. The transformer is designed for a 25-nanosecond pulse. Therefore, the input signal to Q1 must be limited to that length of time. This signal is transmitted down the coaxial cable as a wavefront of voltage and current much the same as the twisted pair case. However, the receiving circuit includes a terminating shunt resistor, which prevents any reflecting waves.

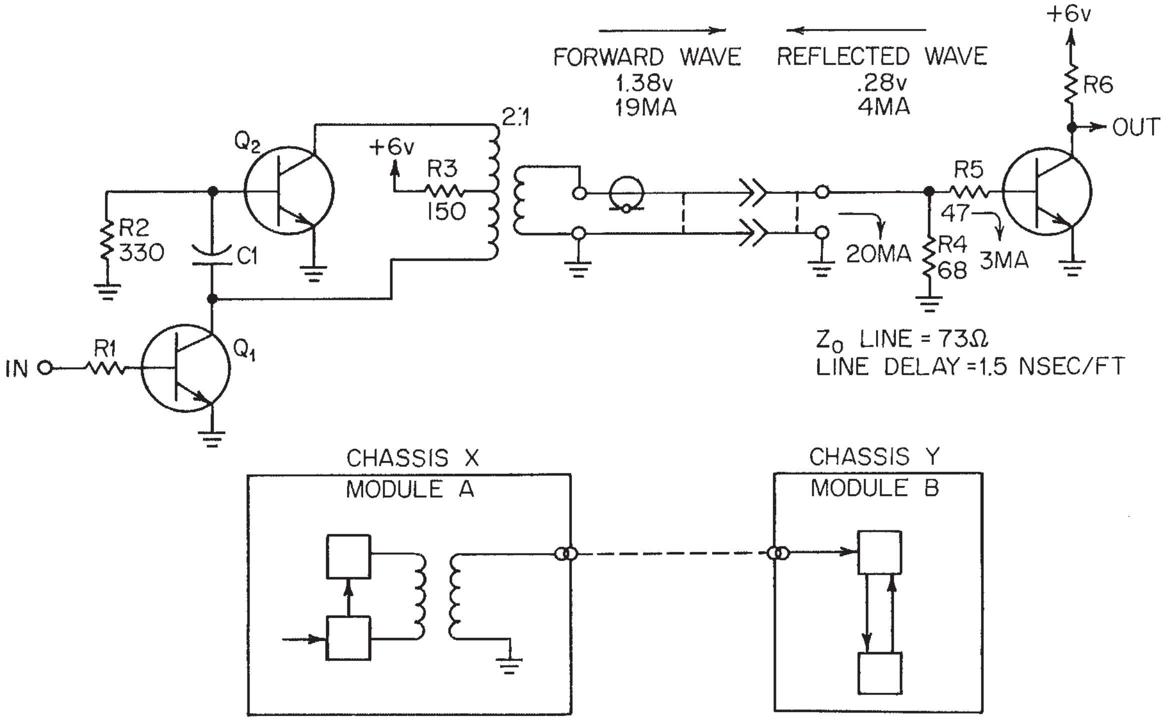

FORWARD WAVE
1.38v
19MA
REFLECTED WAVE
.28v
4MA
+6v
R6
OUT
2:1
Q2
+6v
R3
150
R2
330
C1
R5
47
20MA
R4
68
3MA
Z0 LINE = 73Ω
LINE DELAY =1.5 NSEC/FT
IN
R1
Q1
CHASSIS X
MODULE A
CHASSIS Y
MODULE B

FIGURE 21 Coaxial cable driver.

When Q1 is turned off, the positive-going voltage signal is coupled through the capacitor C1 to the base of Q2, turning it on. Collector-emitter current flowing in Q2 also flows in the second half of the primary winding of the transformer, in an equal but opposite polarity to the first 25 nanosecond pulse. The values of C1 and R are so chosen as to hold Q2 on for 25 nanoseconds, producing a complete cycle of positive and negative signal through the transformer. This second negative polarity pulse produces no effect on

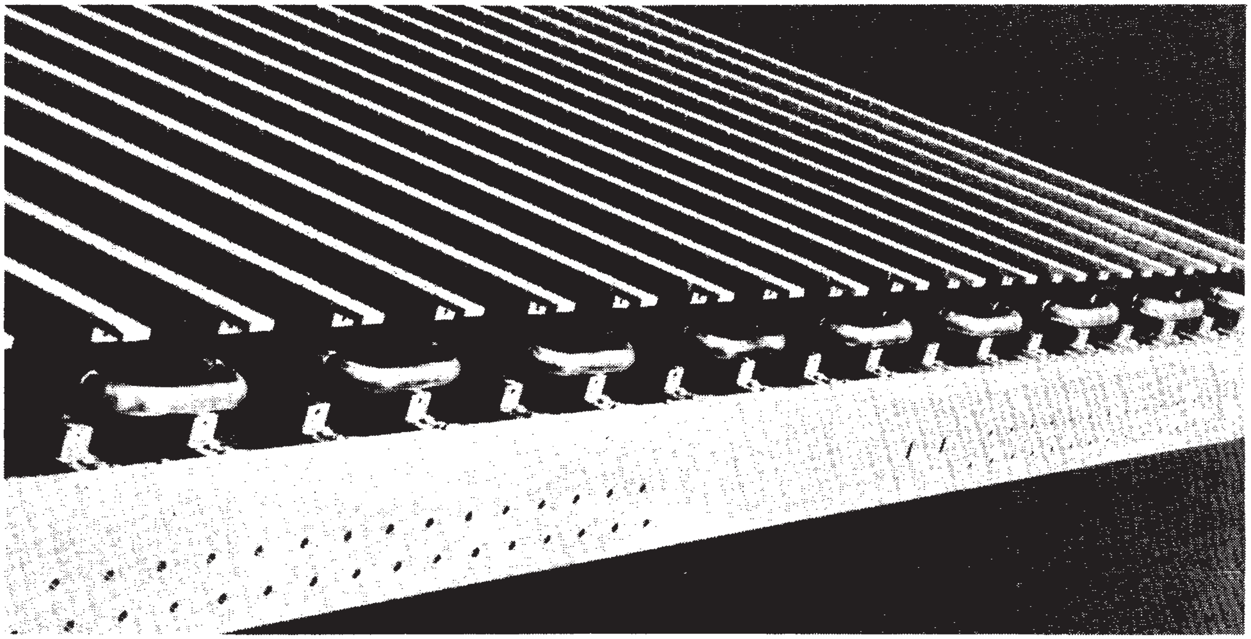


FIGURE 22

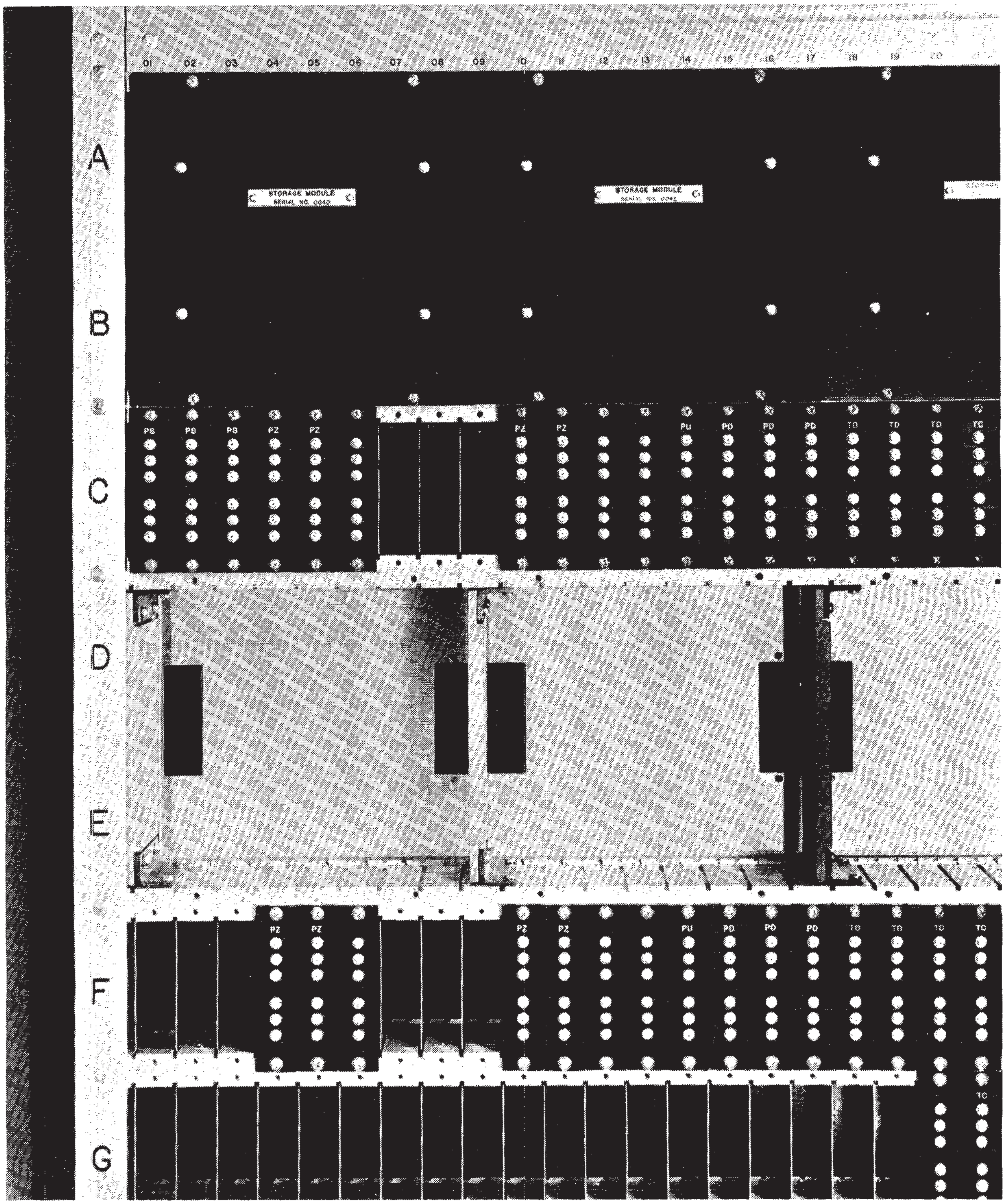

01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
A
STORAGE MODULE
STORAGE MODULE
B
C
PS
PS
PS
PZ
PZ
PZ
PZ
PU
PD
PD
PD
TD
TD
TD
TC
D
E
F
PZ
PZ
PZ
PZ
PU
PD
PD
PD
TD
TD
TD
TC
TC
G

FIGURE 23

the receiving transistor. However, it effectively equalizes the magnetic state of the transformer, removing any "burst" effects.

Only when a "1" is to be transmitted is a pulse sent through the coaxial cable. A new signal can be transmitted every 100 nanoseconds, previously defined as a MINOR CYCLE. Time of propagation in the coaxial cable is 1.5 nanoseconds per foot. Cables used from chassis to chassis are standardized at 10 feet, resulting in a time interval of very close to 25 nanoseconds from the beginning of the input signal to Q1 and the beginning of the output signal from Q3.

All uses of this coaxial transmission within the 6600 Computer involve a synchronous pulse derived directly from the computer's clock oscillator. It is of special interest that the clock itself is also transmitted throughout the system using the coaxial transmission.

E. PACKAGING

From the time that the transistor became available for computer design, packaging methods have become increasingly important. A particularly difficult set of problems centers around the power distribution and cooling methods. As more and more components are crowded together in large computer systems, it has been necessary to examine alternatives to the traditional air cooling. Similarly, the ability to pack components greatly exceeds the ability to reduce the power expended per logical decision. This can be described as in the chart shown in Figure 24.

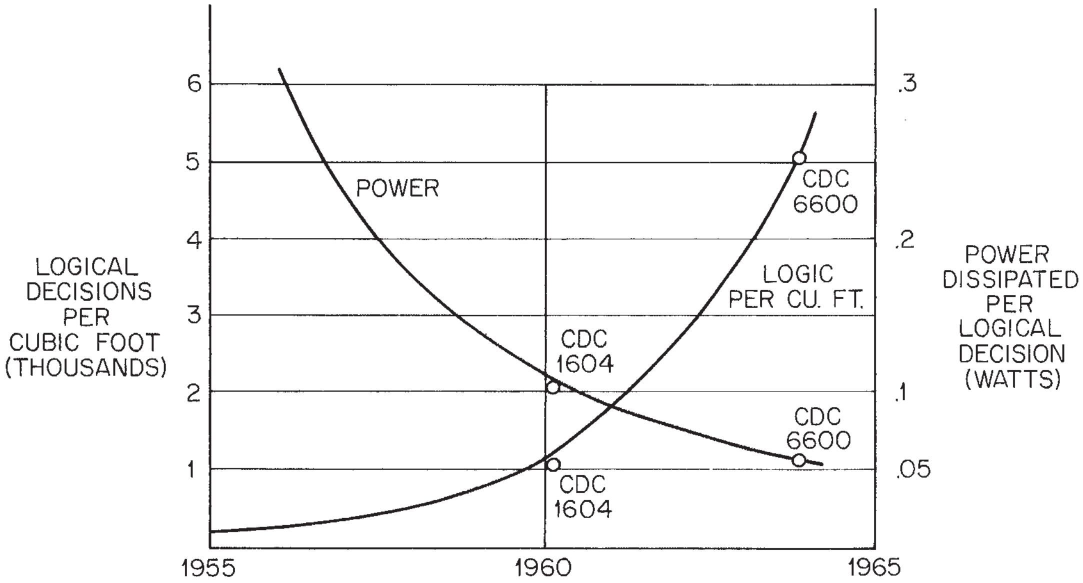


FIGURE 24

The curve of logical decisions per cubic foot ignores space in which no power is being dissipated. What this chart shows is that the packing density increases are causing a net increase in power dissipated per cubic foot, despite the drop in power per logical decision. A word of caution here. This condition exists in the large-scale class of computers, where high-packing density is necessary for achieving increased speeds. For smaller systems this would not be necessary. However, the trends in use of integrated circuits in new designs would appear to be following a similar set of curves.

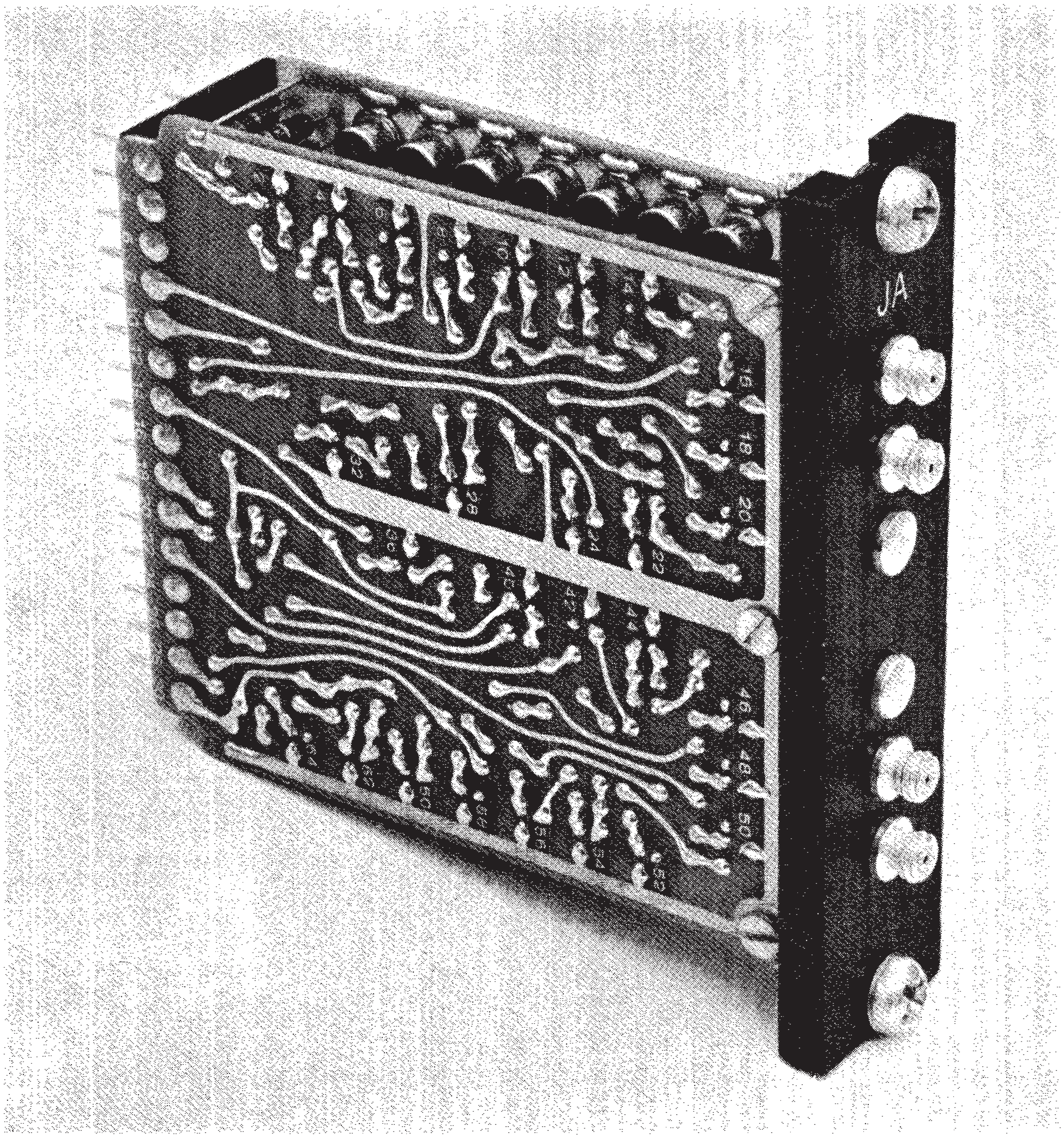
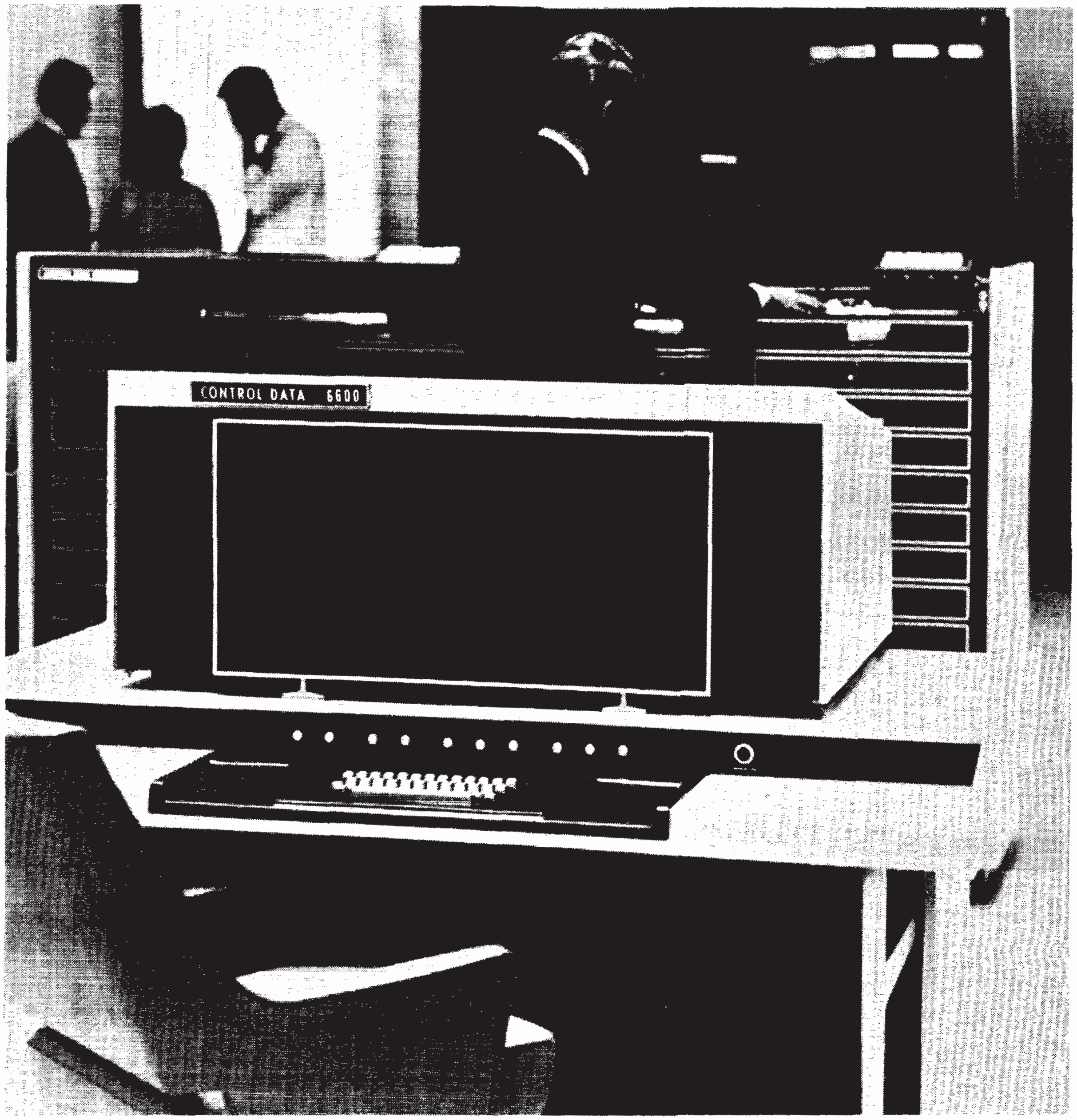


FIGURE 25

Circuits in the 6600 Computer are packaged in modules as shown in Figure 25. Two printed circuit boards are mounted side by side with components mounted in between in a "cordwood" fashion. Transistors in metal cans are mounted on the inside surface of each board; the collector, base, and emitter leads are inserted through holes and soldered on the outer surface to the printed circuit wiring. Resistors are mounted from board to board through holes and soldered also on the outer surface.

A 30-pin connector fastened on one edge of this assembly allows the module to be plugged into the back panel. The module connector mates with a chassis, or back-panel, connector. At the opposite edge of the module assembly is mounted a cover plate containing captive screws. This plate



Operator console with keyboard and display.

connects the module to the chassis mechanically and also forms part of the path for conducting heat from the module. The plate carries an identifying number for the module and up to six test point terminals. These test points are connected to the circuits at points considered most desirable by the designer.

Modules are mounted in a vertical chassis in separate compartments. Compartment side walls, the connector to the back panel, and the module cover plate have a black finish to aid thermal radiation. Horizontal rows of modules are separated and supported by metal bars, similar to shelves. A copper tube is imbedded in each bar and is connected to a Freon refrigeration system. Component heat from each module is carried by conduction through the printed circuit board, module plate, compartment walls, and cold bars. The cold bars are held at a minimum temperature of 60°F. Some control of room humidity is necessary at this temperature in order to prevent dew point condensation.

Central storage modules are similarly pluggable and will be described in a later chapter.

A chassis is capable of holding 756 logic modules. Power from a supply of +6 volts is distributed by a bus bar "ladder" integrated into the chassis structure. Each module has a single lead connecting both ground and the supply voltage. The DC voltage is obtained on the chassis by means of distributed transformer, choke, capacitor, and rectifier elements. Power is

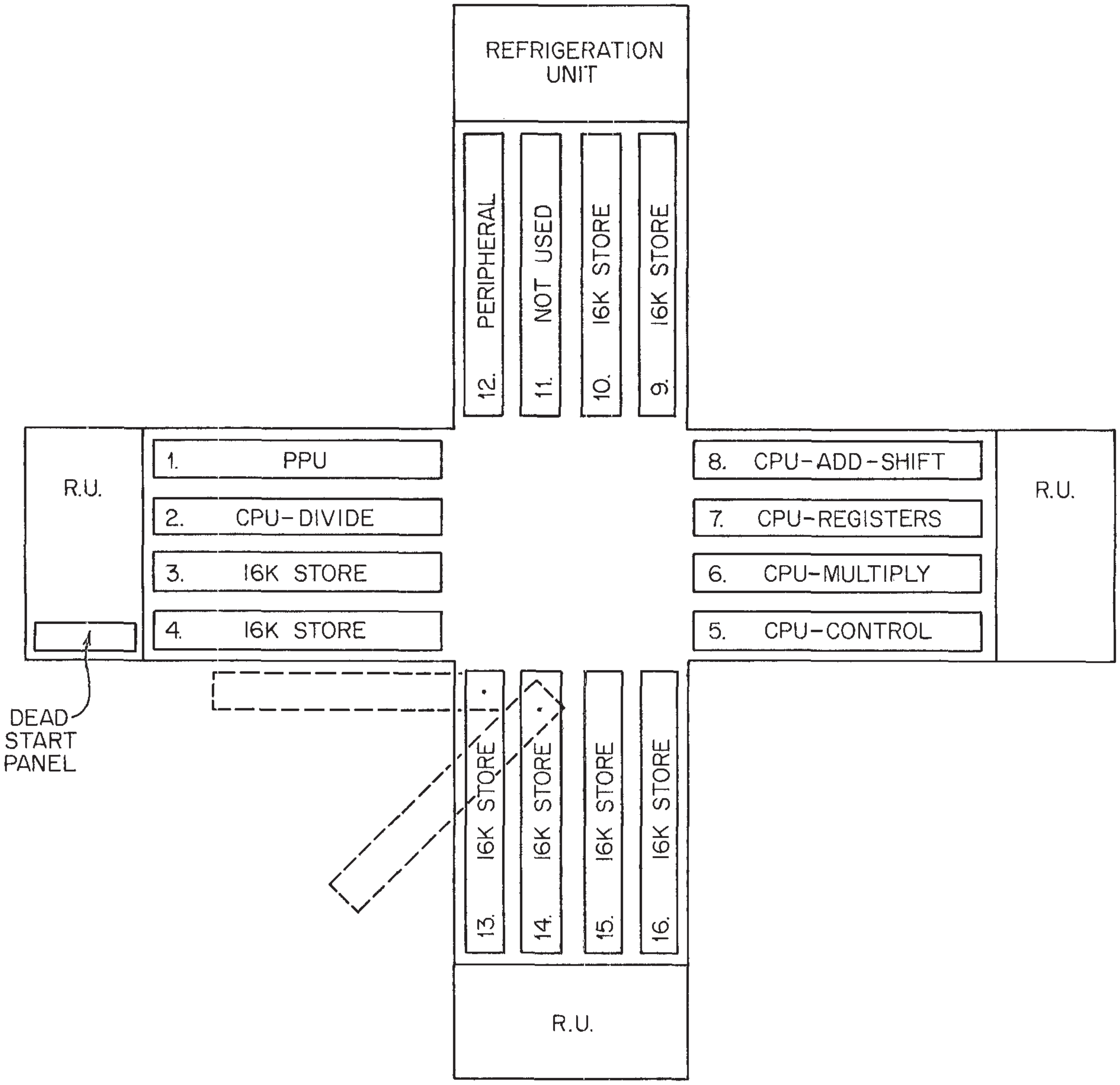

REFRIGERATION UNIT
12. PERIPHERAL
11. NOT USED
10. 16K STORE
9. 16K STORE
R.U.
1. PPU
2. CPU-DIVIDE
3. 16K STORE
4. 16K STORE
8. CPU-ADD-SHIFT
7. CPU-REGISTERS
6. CPU-MULTIPLY
5. CPU-CONTROL
R.U.
DEAD START PANEL
13. 16K STORE
14. 16K STORE
15. 16K STORE
16. 16K STORE
R.U.

FIGURE 26 Cabinet top view.

brought into each chassis through a three-phase, 400-cycle cable from the system motor-alternator. This technique effectively "encloses" the chassis from the point of view of power and ground. The method assumes a relatively well regulated load, a condition which is particularly straightforward with the DCTL circuit.

Chassis are mounted in four cabinet bays, as shown in the top view diagram of Figure 26. The chassis are hinged to swing out as shown in one of the bays. In effect, the chassis are located as if in a cylindrical configuration. The particular value of this method is the minimum cable distances between chassis. All cables are cut to a standard length of ten feet, which includes about four feet within the chassis back-panel area.

CENTRAL STORAGE SYSTEM

IV

Increasingly in large-scale computer systems, the central storage system, or "memory," is the dominating influence on cost and on performance. For fifteen years the ferrite magnetic core has been the basic component used. Other alternatives include magnetic thin film, plated wire and, in early computing days, the Williams tube which was a form of capacitive storage.

Magnetic core storage has ranged from twenty microseconds for a complete read and store operation, using a physically large core, to progressively faster cycles with smaller cores. Techniques for interconnecting these magnetic cores have utilized the coincidence of two or more magnetic fields, usually referred to as coincident current or 3-D. Linear-select methods which do not utilize magnetic coincidence, except for storing, are referred to as 2-D. An interesting intermediate technique is a linear-select method referred to as 2½-D. In general, 3-D is preferred for the smallest storages; 2-D is preferred for the largest storages; 2½-D is found in a narrow intermediate range.

It is a tribute to the magnetic core that many computer people refer to the Central "Core" as synonymous with Central Storage, reflecting the fact that the ferrite core is the component most used for Central Storage.

A. STORAGE MODULE

Within the 6600, Central Storage and each Peripheral Processor use a coincident-current storage module with the following properties.

- 12-bit word length.
- 4096, 12-bit words.
- Read-write cycle of 1000 nanoseconds.

Since Central Storage requires 60-bit words, storage modules are connected as one bank. The module is physically constructed as shown in Figure 27.

The module is an assembly of seven subassemblies, one of which is a three-dimensional matrix of ferrite magnetic cores. The other six are collections of electronic components which "drive" the unit. The entire assembly is built with a connector plate for mounting in the chassis and a cover plate similar to the logic module. In the case of Central Storage, five modules are mounted in a horizontal row of a chassis, taking up the space of two rows of

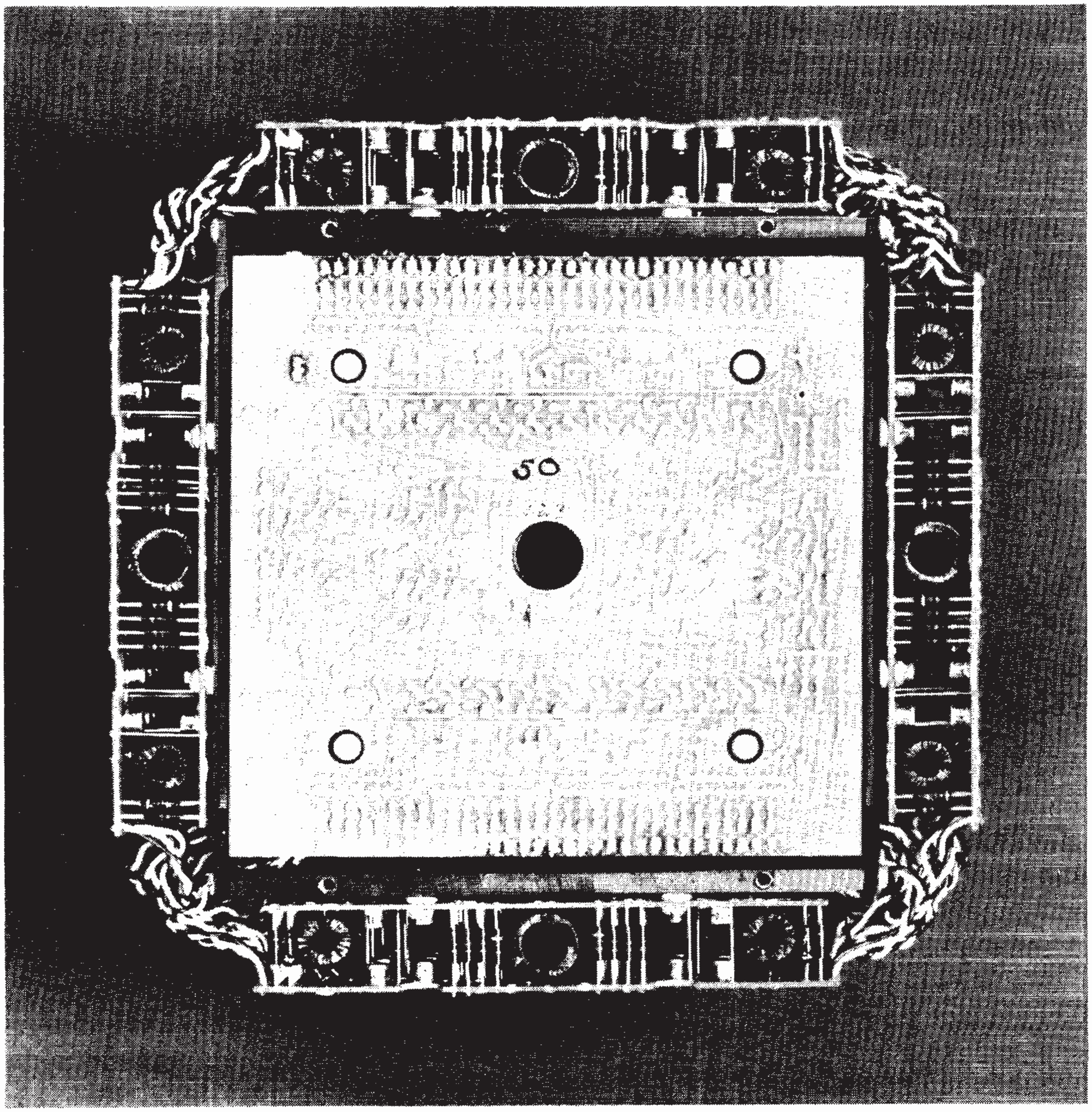


FIGURE 27

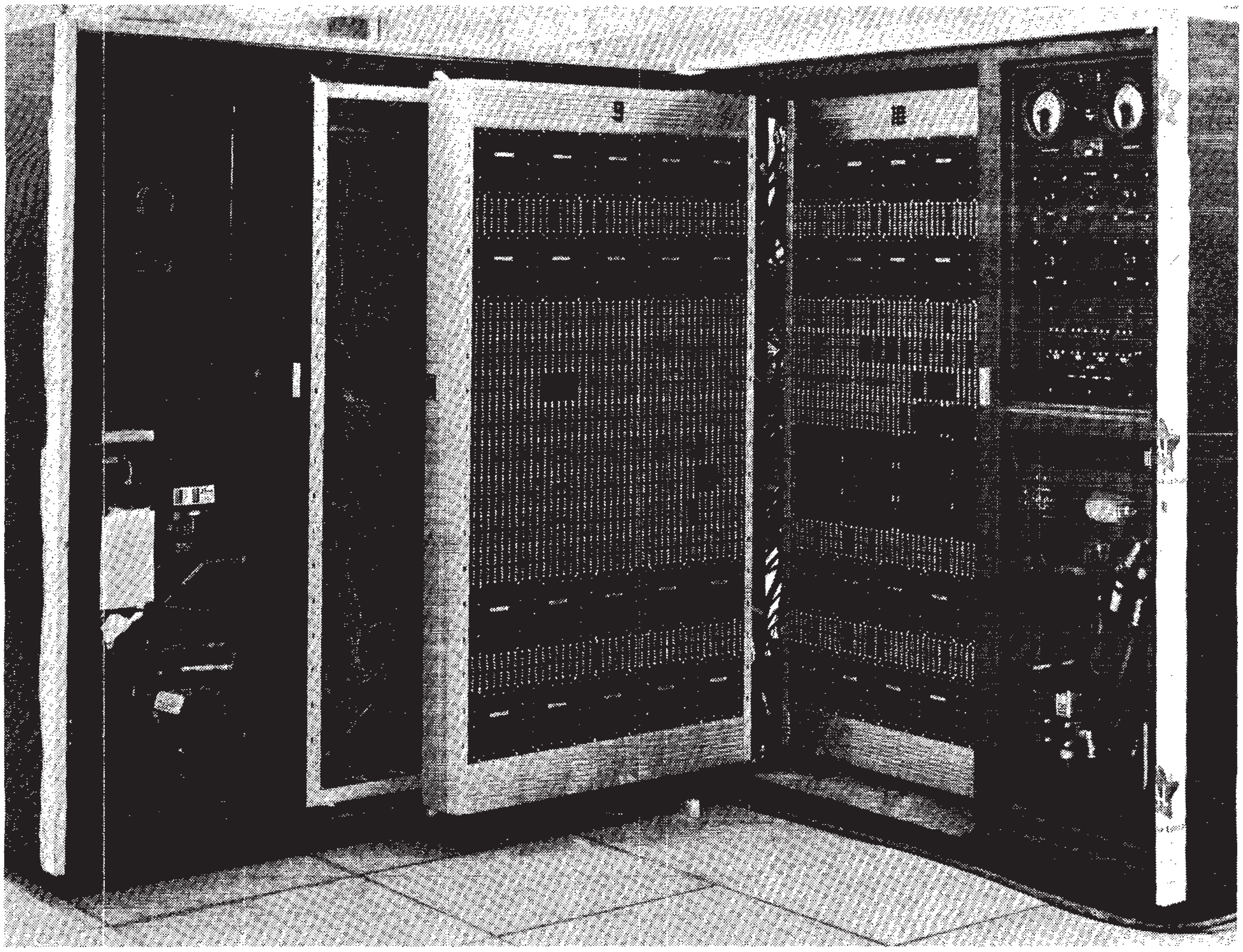


FIGURE 28

logic modules. A single row of logic modules is needed to control this Central Storage bank. Four banks are mounted in each Central Storage chassis, as seen in Figure 28.

Operation of this storage is dependent on a "square-loop" magnetic property of the ferrite magnetic core. A typical representation of this property is shown in Figure 29 (page 40). The point labeled H_C is a measure of the coercivity of the material. A field applied in the H direction causes the magnetic flux density B to change following the shape of this curve. As the field is increased, a point called the saturation point of the material is reached and labeled on the curve B_S. At this point, no amount of additional field will cause any change in the remanent magnetic state B_R following removal of the field.

It is particularly important to a coincident current 3-D storage that the squareness be nearly ideal. For example, the area labeled KNEE is particularly sensitive, as will be seen. This property, squareness, is sensitive to temperature and mechanical stress. Therefore, a physically protected and thermally controlled environment is necessary.

The coincident-current method is based on the coincidence of two fields within a core for a READ or a WRITE operation. These fields are supplied by

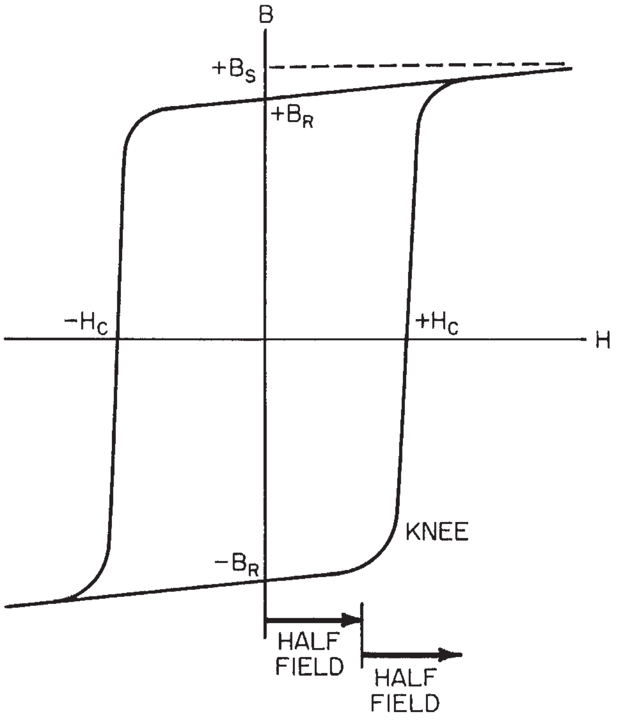


FIGURE 29 Hysteresis loop.

currents flowing in two orthogonal wires passing through the toroidally shaped core. Depending on the direction of the current in these two wires, a positive or a negative field will appear at the core. It can be seen that a "half-current" will produce a "half-field" which can be held just below the KNEE of the B-H curve. Two positive half-currents will produce a full-field which will be substantially above the coercivity of the material as shown in the figure. This example also shows the resulting remanent magnetization following removal of a half-field and a full-field.

It should be clear that two orthogonal wires can be made to select one core from a two-dimensional array. In such a case only one core at the coincidence of the X line and the Y line will experience a full-field. All other cores on the X and Y lines will experience a half-field, while the rest of the cores will remain unaffected by any field.

During a full-field condition the magnetic core will switch states, taking a finite time interval to accomplish the switch. This time interval is 400 nanoseconds in the storage module for the 6600 Computer and is a function of the composition of the ferrite and the dimensions of the core.

A two-dimensional array of cores is shown in Figure 30. The two orthogonal wires X and Y can be seen along with three other wires passing through each core. A diagonal wire is a convenient means for sensing the voltage induced during a core switching operation and is labeled the SENSE, or S wire. The other orthogonal wires are included as a convenient means for counteracting the fields induced by the X and Y lines. These are labeled INHIBIT, or I wires, and effectively allow the array to grow from two dimensions to three. Figure 31 (page 42) diagrams the method used.

In this diagram one X drive line and one Y drive line are shown. A separate inhibit line is used for each bit or layer in the third dimension, while the X and Y lines thread through the whole array as shown. Each "plane" is a two-dimensional array of 4096 bits. There is a total of twelve planes in a module, making up the twelve bits of word length. Bit control is accomplished through the X inhibit and Y inhibit wires on each plane.

A typical readout is accomplished by pulsing the X and Y lines with half currents in a direction such as to produce a "positive" full-field in each of the twelve cores corresponding to the selected address. One can easily predict the switching behavior of a core at a selected address for the two cases of interest. With a "1" previously stored in the core, a negative remanent state $-B_R$ exists before the readout. The full readout field causes the core to follow the B-H curve to the right and upward, finally coming to rest at $+B_R$. The total magnetic flux change is represented by the excursion from $-B_R$ to $+B_R$, resulting in a "one" signal readout on the SENSE winding. If this signal is sampled and stored, the WRITE cycle of the storage unit can be used to replace the "1" in the core. A WRITE cycle follows the READ cycle by

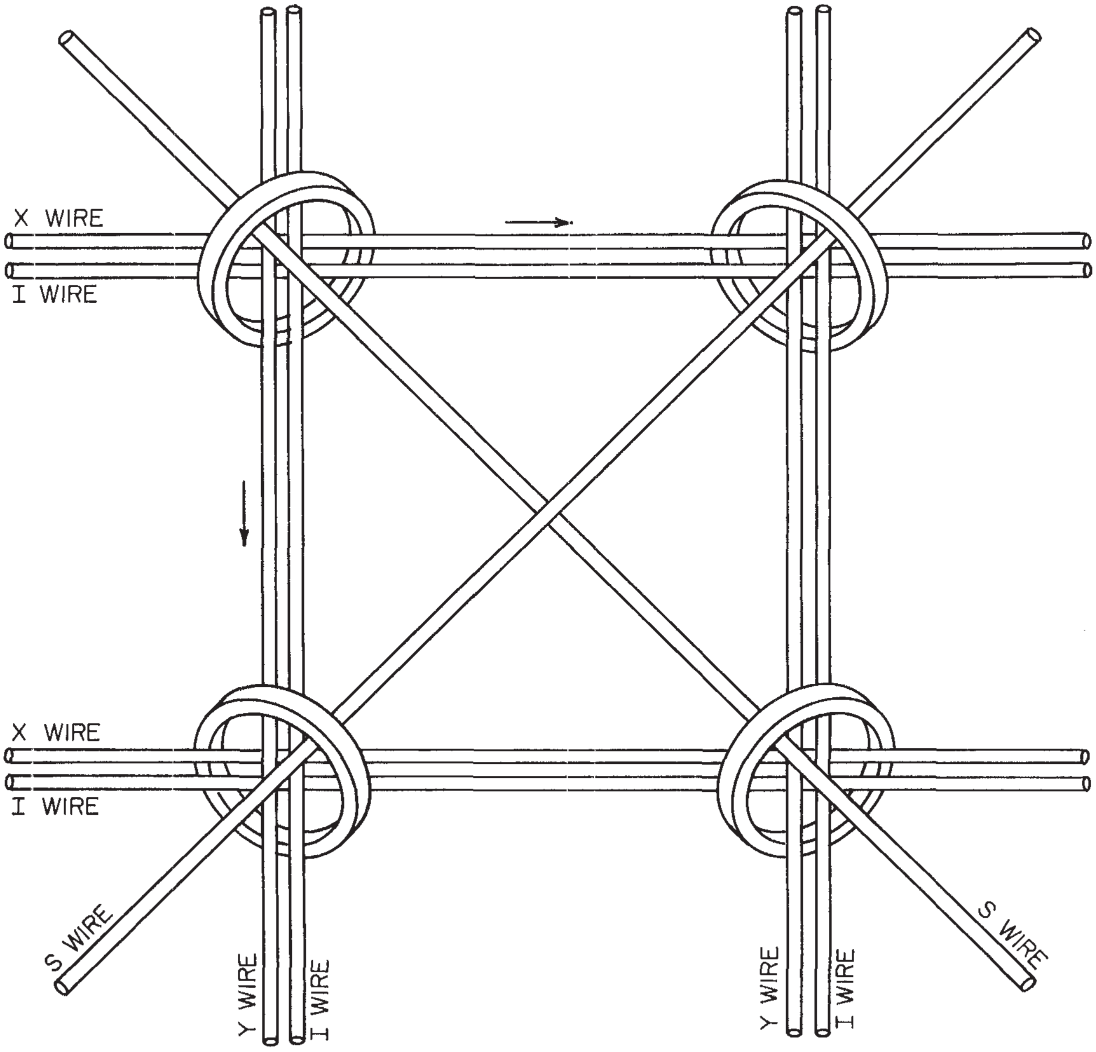


FIGURE 30

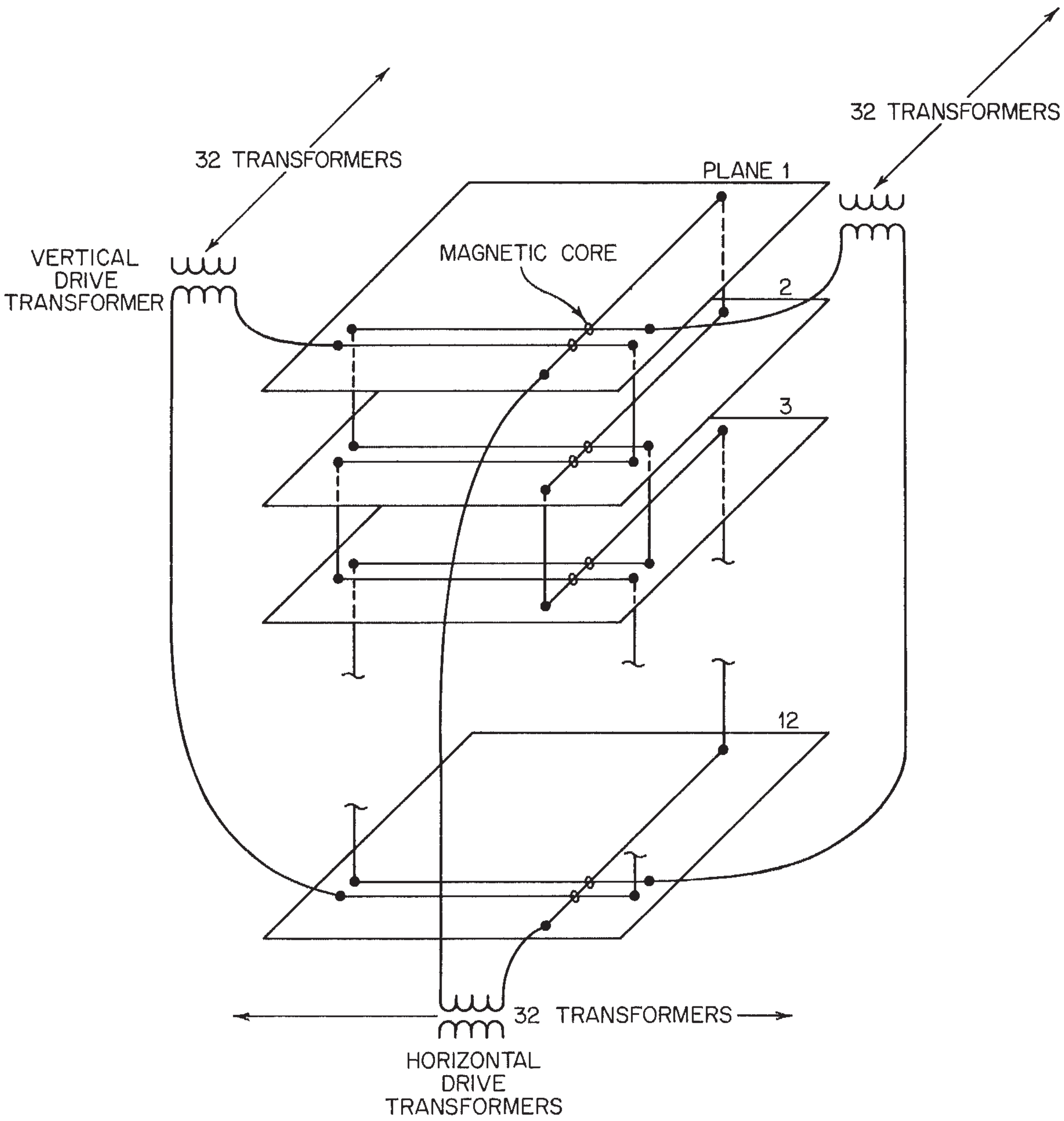

32 TRANSFORMERS
32 TRANSFORMERS
PLANE 1
VERTICAL DRIVE TRANSFORMER
MAGNETIC CORE
2
3
12
32 TRANSFORMERS
HORIZONTAL DRIVE TRANSFORMERS

FIGURE 31 Drive line connections.

causing the X and Y lines to produce a "negative" full-field in each of the twelve planes, again at the selected address. The result obviously is to restore the cores to their original magnetic remanent state.

The second case of interest is found with the initial magnetic remanent state at $+B_R$ representing a stored "zero." The positive full-field during the READ cycle causes very little actual flux change, resulting in a very small signal. During the following WRITE cycle, the INHIBIT windings are energized by a positive half-field counteracting the effect of the full negative X-Y drive. This is shown in the diagram of Figure 32.

This operation is shown in Figure 33 in the form of signal waveforms for drive and inhibit currents. The type of storage just described is known as "destructive readout" or DRO storage because a single READ requires a following RESTORE to retain the data.

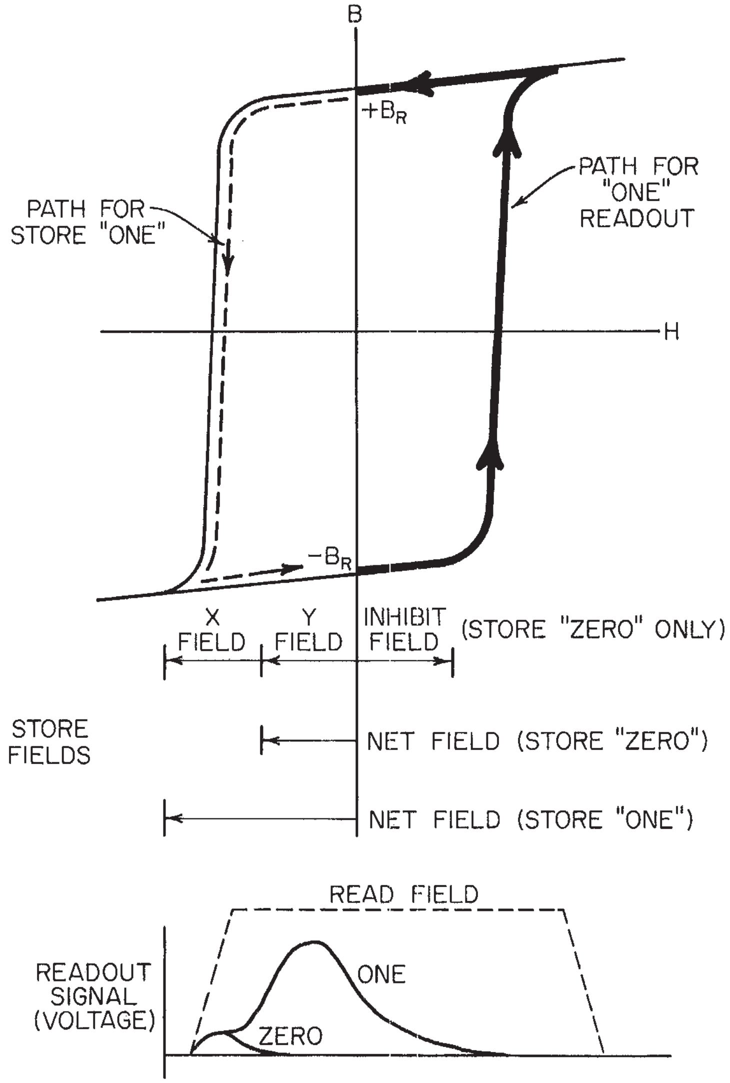

B
+B_R
PATH FOR "ONE" READOUT
PATH FOR STORE "ONE"
H
−B_R
X FIELD
Y FIELD
INHIBIT FIELD
(STORE "ZERO" ONLY)
STORE FIELDS
NET FIELD (STORE "ZERO")
NET FIELD (STORE "ONE")
READ FIELD
READOUT SIGNAL (VOLTAGE)
ONE
ZERO

FIGURE 32 Read and store.

All storage modules in the Central Storage and in the Peripheral Processors utilize an identical cycle. This cycle is controlled by a sequence control for each Central Storage bank and by the Peripheral Timing Control to be described in a later chapter. Two types of Storage References can be made, a READ/RESTORE and a CLEAR/STORE with a special EXCHANGE

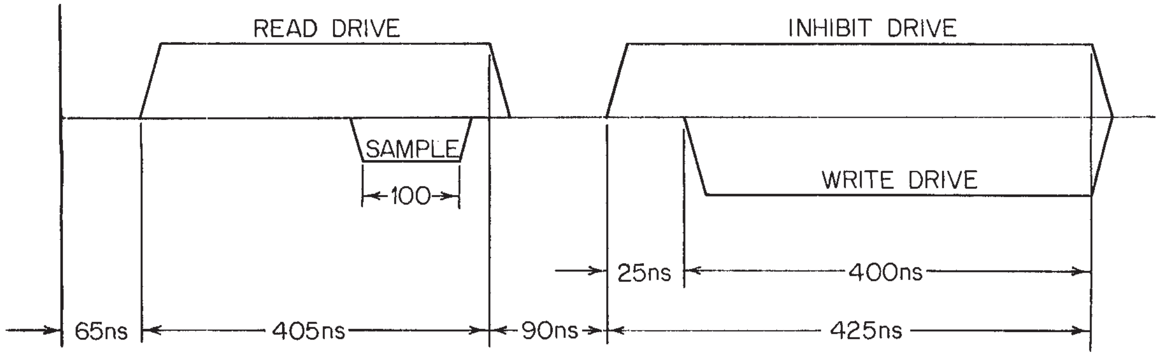

READ DRIVE
SAMPLE
100
INHIBIT DRIVE
WRITE DRIVE
25ns
400ns
65ns
405ns
90ns
425ns

FIGURE 33

reference to be described later. In any case, a full 1000 nanoseconds are always used for any reference.

The storage module block diagram of Figure 34 shows the logic needed. From this block diagram it can be seen that data can be read out and new data stored in a given address in one cycle. In the case of Central Storage, five Storage Modules are used with a single address and with a 60-bit Z register and 60 sense amplifiers.

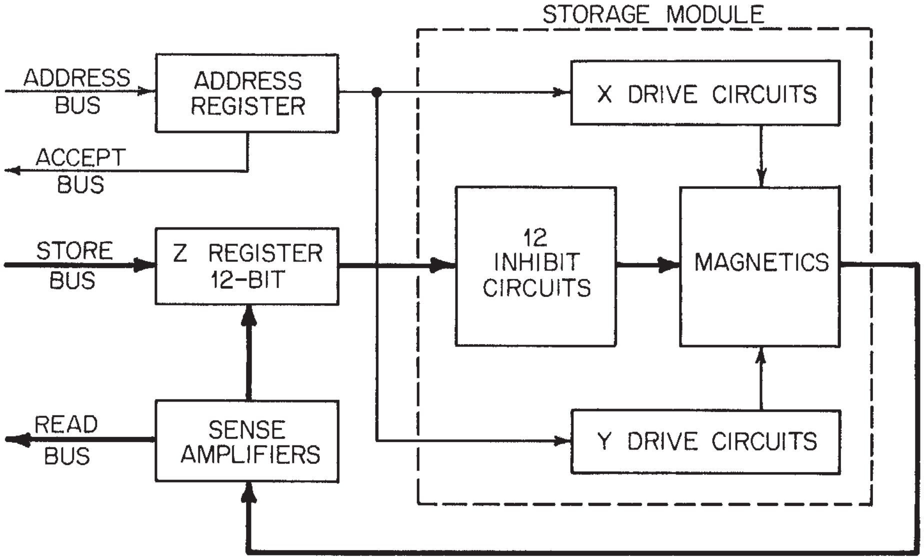

STORAGE MODULE
ADDRESS BUS
ADDRESS REGISTER
ACCEPT BUS
X DRIVE CIRCUITS
STORE BUS
Z REGISTER 12-BIT
12 INHIBIT CIRCUITS
MAGNETICS
READ BUS
SENSE AMPLIFIERS
Y DRIVE CIRCUITS

FIGURE 34

B. THEORY OF INTERLEAVED STORAGE

It has not been possible to construct an economically reasonable single-level central storage. There is also a limit on size versus performance, arising from the proportion of time spent getting to and from storage as against the time of a storage reference within the unit itself.

Both cost and performance combine to force a system of primary and secondary storage; in other words, a hierarchy.

In the following discussion of an optimum solution to the storage system organization, it is well to remember that the solution must hold for the whole system. A denial of the existence of the hierarchy can be countered by the realization that small central storage can be very fast, but large cannot. Obviously, either the central storage will be too small at one extreme or too slow at the other.

TRANSFER BETWEEN LEVELS

With a beginning assumption of only two levels of storage hierarchy, a first consideration is the transfer between the two levels. Unquestionably the two levels will have different properties with the advantage going to performance (speed) at the higher level and with the advantage going to cost at the lower level. It should be clear that this will produce a secondary level module much larger than the primary level.

Assuming the secondary storage is a rotating device such as a magnetic drum or disk, there are other differences in properties between levels. The secondary storage will have three types of timing considerations related to the transfer of data between levels. These are listed here.

1. Positioning time—this refers to any mechanical action to place the recording or reading head on the desired track.
2. Latency time—assuming a rotating mechanism, this refers to a portion of one rotation to reach the desired angular position on the track.
3. Transfer rate—this refers to the rate of transferring data to or from the storage unit.

As positioning and latency time get very large, a number of strategies must be employed to prevent major inefficiencies. Such strategies can include:

1. a number of independent secondary storages to overlap positioning times;
2. queuing methods to take advantage of transferring between levels on the basis of whatever is "closest";
3. insertion of an intermediate level of storage to smooth the secondary storage traffic;
4. adoption of a time-shared usage of the central processor in the hope of providing job overlap.

The optimum choice of strategy is extremely dependent on the type of job or jobs being executed and the size of the transfers between levels. It should be apparent that as the secondary references become more random as to position, the first strategy above of independent secondary storages becomes more necessary.

The transfer rate of data flowing between levels should be balanced with the type of secondary storage used. However, as the queuing strategy or time-shared technique improves, the transfer rate becomes quite important. See Chapter VII for Disk Storage Unit.

PRIMARY STORAGE

At the top of the storage hierarchy the question of the optimization is very complex. This storage must provide peak performance with the processor; it must provide for the transfer to and from the secondary storage with minimum loss to the processing job; it must provide for the data transfers to and from the external devices.

Looking at the processor performance first, a major limitation is the time taken in the transfer of each operand or result between the processor and the primary storage. Assuming the processor can be infinitely fast, each storage reference required in the process can cost a storage cycle time.

It is obvious that one should attempt to reduce the number of actual references to storage. Some possible strategies would be:

1. take more data per reference in the hope that more than one element of data can be used;
2. utilize a small scratch pad memory for intermediate or partial results of computation which need not transfer between the processor and storage;
3. remove extraneous references to storage for the housekeeping associated with the processor program, either by microprograms with the scratch pad or by special instruction hardware.

The necessary references to storage from the processor may be aided by other strategies.

1. Split primary storage into independent units, each servicing a separate processor.
2. Split primary storage and provide overlap conditions to hide portions of the storage cycle.
3. Interleave many banks of primary storage in order to increase the speed of "burst" transfers between the processor and storage, and between levels of storage.

For random referencing of primary storage and with the ability to accomplish an overlap, the more independent banks of storage the better. Of course, as the number of banks increases, the processor speed can decrease due to increased line length and increased distribution and switching logic.

For burst transfers between the processor and storage a reasonable assumption can be made that the transfer is made on a series of consecutive addresses. In that case, many banks of storage with the ordering of consecutive addresses assigned to consecutive banks is a possibility, such as:

Word Address	Bank
—	—
—	—
6713	3
6714	4
6715	5
—	—
—	—
—	—

This sort of interleaving is not incompatible with random referencing requirements since all that is needed then is many banks without special regard to addressing order.

SYNCHRONISM

In a primary storage system two methods may be employed to connect each storage unit, namely, synchronous connection and asynchronous connection. The synchronous connection provides a clock line between all units and specifies a fixed time interval for control and data transfer events. The asynchronous connection requires no clock line and does not specify a time interval.

The advantage of the asynchronous connection falls in the category of modularity, or simple expansion, without regard to distance or other time variables.

If no attempt is made to overlap storage operations, or if overlap is limited to Read and Store overlap, then the asynchronous technique has the flexibility advantage. However, as any attempt is made to increase the overlap, the timing uncertainty of the asynchronous connection is an obstacle.

SYNCHRONOUS OVERLAP

The following describes a method of connecting a number of banks of storage in an interleaved synchronous manner to achieve a high burst rate and a high degree of random overlap. Assume these properties:

1. a Read-Store reference takes one MAJOR CYCLE;
2. the storage bus can transfer one word in one MINOR CYCLE;
3. storage accessing agent can deliver a new address every minor cycle;
4. after the address is delivered, a data word will be transferred after a fixed number of minor cycles.

The storage system can now take advantage of the predictable nature of the synchronous connection. Many storage references can be made at minor cycle intervals with the resultant data transfer at fixed following intervals. Conflict in the use of a storage unit can be detected in a predictable manner in advance of actual data transfer. The strategy to be employed in the presence of a conflict may vary with the kind of reference.

This method of overlap is entirely legitimate for random references and for mixing several referencing units. It is also very powerful for burst transfers where a single control instruction can set up a stream of data with data words transferring at minor cycle intervals.

C. STUNT BOX

Based on a synchronous interleaved storage system as described above, the 6600 Computer makes use of its high degree of overlap. The mechanism

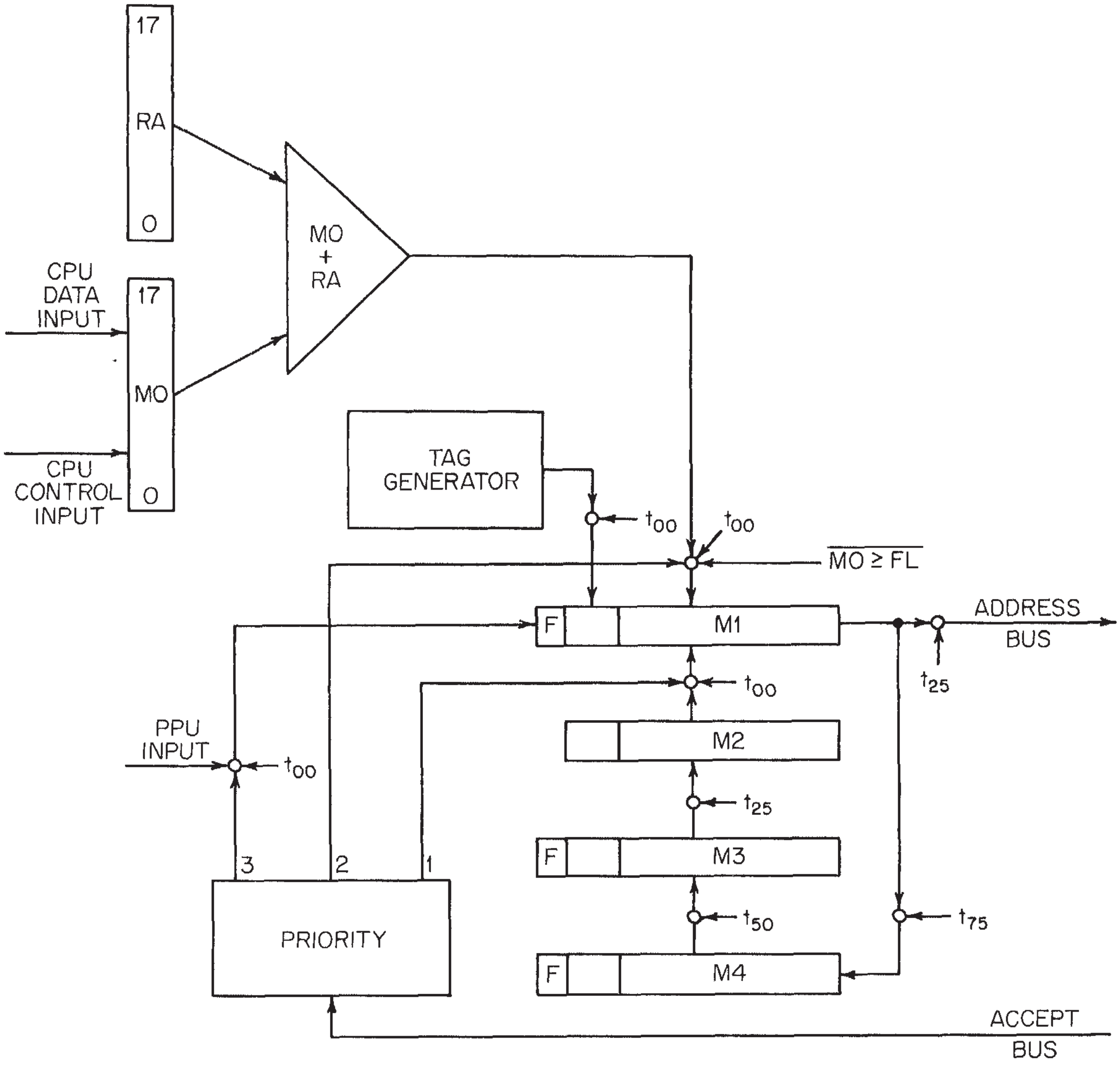

17
RA
0
MO
+
RA
CPU
DATA
INPUT
17
MO
CPU
CONTROL
INPUT
0
TAG
GENERATOR
t00
t00
MO ≥ FL
F
M1
ADDRESS
BUS
t00
t25
M2
PPU
INPUT
t00
t25
3
2
1
F
M3
PRIORITY
t50
t75
F
M4
ACCEPT
BUS

FIGURE 35

for referencing control storage is called the STUNT BOX. This unit is shown in block diagram form in Figure 35.

The STUNT BOX contains three main parts:

- hopper,
- priority network,
- tag generator and distributor.

The *hopper* is an assembly of registers used to retain storage reference information until any storage conflicts are resolved. In principle this allows a new storage address to be delivered every minor cycle with any "rejected" addresses to be reissued repeatedly until accepted.

Assuming an empty hopper, a storage address is entered in register M1 from one of the sources, the Central Processor or Peripheral Processors. This entry is made under control of the priority network, with the time of entry at t00. As described in a previous chapter, this time is the leading edge of a 25-nanosecond clock pulse and is repeated every minor cycle, or 100 nanoseconds. At the time of this entry a set of tags is also entered, which fully identify the nature of that particular storage reference. The storage address is sent immediately to the storage units on the storage address bus, at clock time t25.

The hopper is designed to allow the information entered in register M1 to circulate through the other registers and return to M1. A 75-nanosecond time interval exists between each register which results in a total recirculation time of 300 nanoseconds. For example:

t00 —Enter M1
t25 —Address to Storage Address Bus
t75 —M1 to M4
t150—M4 to M3
t225—M3 to M2
t300—M2 to M1 (if not accepted)

In each storage bank the lower five bits of the address are examined for a match. If the bank whose number matches is not busy, an ACCEPT signal is returned on the ACCEPT Bus to the Stunt Box. This indicates that no conflict exists, and the storage cycle has been initiated in the bank. The ACCEPT signal reaches the Stunt Box in time to disable the path from M2 to M1 and to remove that entry from the priority network. As a result, a new address may be entered. If, however, the selected storage bank was busy, no ACCEPT signal would be sent. In that case, the priority network gives top priority to the recirculation path, thereby causing new addresses to be held up for one minor cycle. It should be obvious that the hopper can hold three addresses recirculating, effectively blocking any new addresses. As each recirculating address is accepted, a new address can be entered.

This method of handling storage accesses has the considerable value of preventing unnecessary bottlenecks. One source of addresses will not block another source except as each may call for the same storage bank. Knowing the recirculation time of 300 nanoseconds and the storage cycle time of 1000 nanoseconds, an interesting case is a series of consecutive references to the same bank. Shown in Figure 36 is a worst-case condition of references filling

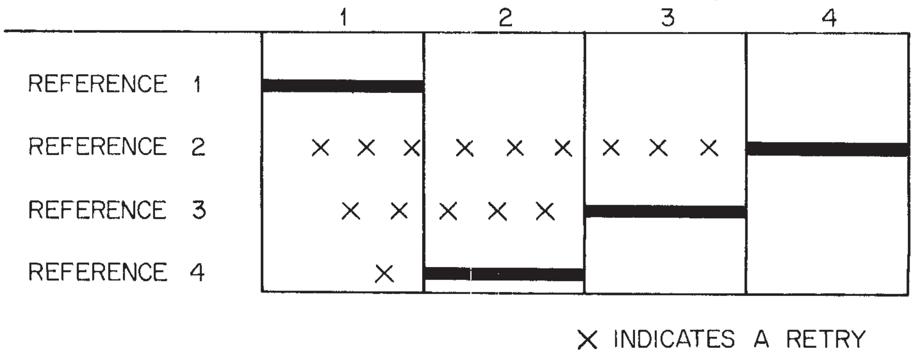


FIGURE 36

the hopper and being recirculated out of order. The maximum case is shown for reference 2 which was delayed by two major cycles. Note that each re-try by reference 2 prevents any new address from making the delay any longer. This points up a fact about this mechanism, that holding three entries rather than two, four, or more, will prevent "permanent" recirculation.

The *priority network* includes several address sources under the central processor and the peripheral processor headings. From the central processor, two classes of storage reference are seen:

1. instruction fetch, and
2. read and store operands.

Instruction addresses are obtained from the central processor program address register (P), whereas read or store operand addresses are obtained from one of two INCREMENT functional units. In the case of simultaneous references from these two classes, priority is given to the operand address. When one of these is entered into register MO (see Figure 35), a request for priority 2 is made in the priority network. The priority network also accounts for the presence of a recirculation priority, rejects for mixed read and write in the hopper, and also rejects for out-of-bounds address.

The mixed read and write reject is a special case for the central processor references in order to prevent the out-of-order recirculation properties of the Stunt Box from causing out-of-order operations on a single storage location.

The out-of-bounds address is determined by testing the address in register MO against the field length (FL) which is established for each program by the operating system. Also, note that each central processor address is increased by the reference address (RA) also supplied by the operating system. These will be described in more detail in later chapters.

In summary, the priority #2 for central processor references is made up of:

1. address in MO, AND
2. no recirculation requirement, AND
3. no mixed read and write, AND
4. address in MO less than FL.

Similarly, the peripheral processor addresses are entered from several sources. There are, of course, ten PPU's of which only one can reference the Central Storage at one time. The reference from the PPU is also one of three types, a read central, write central, or initiate exchange. This last causes the central processor to halt and perform an exchange of the contents of its registers with a sixteen-word storage block beginning at the address specified by the PPU.

PPU reads and writes are handled as individual 60-bit word references requiring an address each. For the Exchange, however, only the starting address of the exchange jump package is entered. A counter (exchange address counter EAK) is used for the remaining storage addresses. This counter is placed in the address path from the PPU's to the Stunt Box for convenience and is deactivated for reads and writes.

The *tag generator* is designed around a six-bit tag. Three bits are used to identify registers for the exchange jump and central read or write. The other bits are assigned to control, read, write, or exchange for the various sources. Table III shows the net combination, using two octal digits for the six-bit tags.

TABLE III Hopper Tags

00	Peripheral READ
10	CPU-instruction fetch
11	CPU read to X1
12	CPU read to X2
13	CPU read to X3
14	CPU read to X4
15	CPU read to X5
40	Peripheral Write
50	Return Jump or Error Stop
56	CPU write X6
57	CPU write X7
60	Exchange P, A0
61	Exchange RA, A1, B1
62	Exchange FL, A2, B2
63	Exchange EM, A3, B3
64	Exchange RA ecs, A4, B4
65	Exchange FL ecs, A5, B5
66	Exchange A6, B6
67	Exchange A7, B7
70	Exchange X0
71	Exchange X1
72	Exchange X2
73	Exchange X3
74	Exchange X4
75	Exchange X5
76	Exchange X6
77	Exchange X7

These tags are circulated with the addresses in the hopper with tags distributed into control logic as needed. Some tags initiate operations which are independent of the ACCEPT. Other operations which are dependent on the ACCEPT are initiated by delayed copies of the tags. Therefore, once in the Stunt Box each address tends to control itself.

D. STORAGE BUS SYSTEM

Central Storage in the Control Data 6600 is constructed on eight independent chassis, necessitating a distribution system. Some discussion was given earlier about the Storage Address Bus. In actuality, the Stunt Box delivers addresses directly to each of the eight chassis. Each chassis contains four banks of 4096 60-bit words. In the case of a 65,536 word central storage, only four chassis are used, and the distribution system is accordingly reduced.

A comprehensive block diagram of the storage data distribution system is given in Figure 37 (page 52).

Since there are several sources and destinations for the Central Storage references and since there are physically independent banks of central stor-

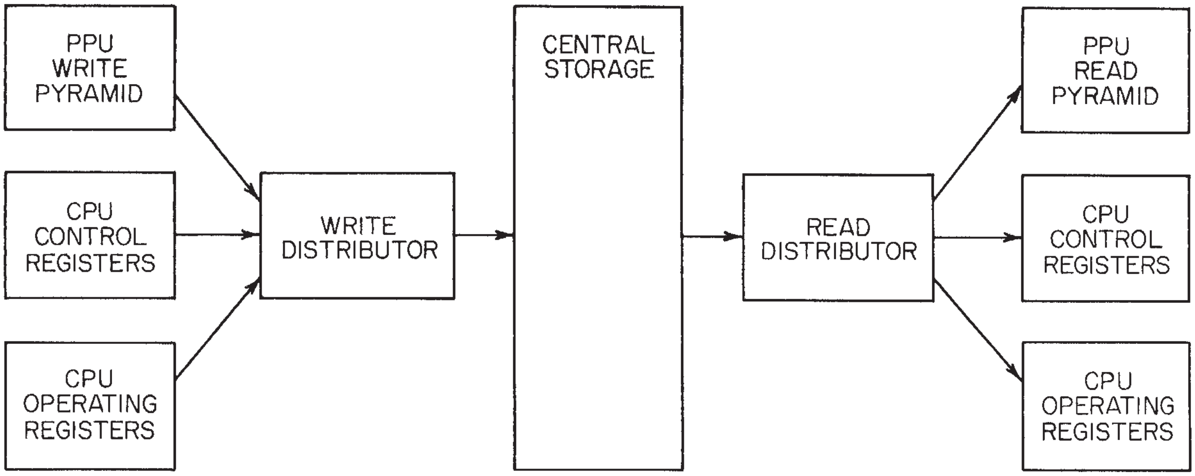

PPU WRITE PYRAMID
CPU CONTROL REGISTERS
CPU OPERATING REGISTERS
WRITE DISTRIBUTOR
CENTRAL STORAGE
READ DISTRIBUTOR
PPU READ PYRAMID
CPU CONTROL REGISTERS
CPU OPERATING REGISTERS

FIGURE 37

age, two *distributors* are needed. These represent intermediate points for concentrating and distributing between sources and destinations.

When a PPU reference to Central Storage is made, the address is delivered to the Stunt Box. If the reference is a Write Central, the 60-bit word to be stored is held in a register associated with the Write Distributor. When that reference is accepted, the tag associated with the address in the Stunt Box hopper causes the data word to be transmitted through the Write Distributor to Central Storage. Other references are similarly handled.

Data words transfer through this system in such a way that a word to be stored arrives at the chassis just after the word read out of the address has been transmitted. This can be seen in Figure 38.

It is possible to transfer data in each direction through this system, an essential for the Exchange operation. Since it takes a finite time to move data words through the write distributor and the read distributor, the controls derived from the Stunt Box tags are spaced out in time over a wide interval.

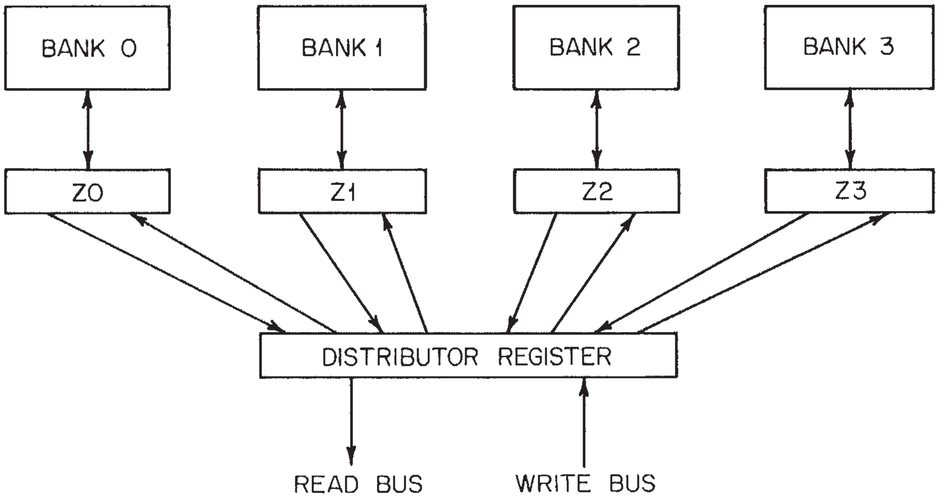

BANK 0
BANK 1
BANK 2
BANK 3
Z0
Z1
Z2
Z3
DISTRIBUTOR REGISTER
READ BUS
WRITE BUS

FIGURE 38

Looking at the read-restore "loop" of a storage bank, it can be seen that data words are read out to the Z register and then to the common distribution register. The data word to be restored is then returned to the Z register. This operation is somewhat artificial since the Z register already has the data word. However, this is a convenient control procedure. Data read out of any of the four banks is transmitted on the READ Bus on all storage references. All eight chassis are connected to the Read Distributor which, in turn, transmits to the correct destination. In the case of a STORE operation, the readout data is effectively dropped at the Read Distributor.

E. EXTENDED CORE STORAGE

Following the ideas expressed in an earlier section, the Extended Core Storage is inserted in the Storage hierarchy to "smooth" the traffic between storage levels. The connection is direct "core to core" for maximum transfer rate. The CPU contains two instructions for transferring between Central Storage and Extended Core Storage, one for READ, and one for STORE. The CPU may direct a transfer of any number of 60-bit words from one to the maximum permitted by field length assignments. Extended Core Storage addresses in the CPU are relocated by a reference address (RA ecs) similar to, but separate from, the reference address for Central Storage. Also, a field length (FL ecs) is assigned by the operating system. These provide a powerful mechanism for relocation and protection in both levels of storage. For descriptive purposes, these are shown in Figure 39.

The CPU addresses in a relative space starting at zero for both storages.

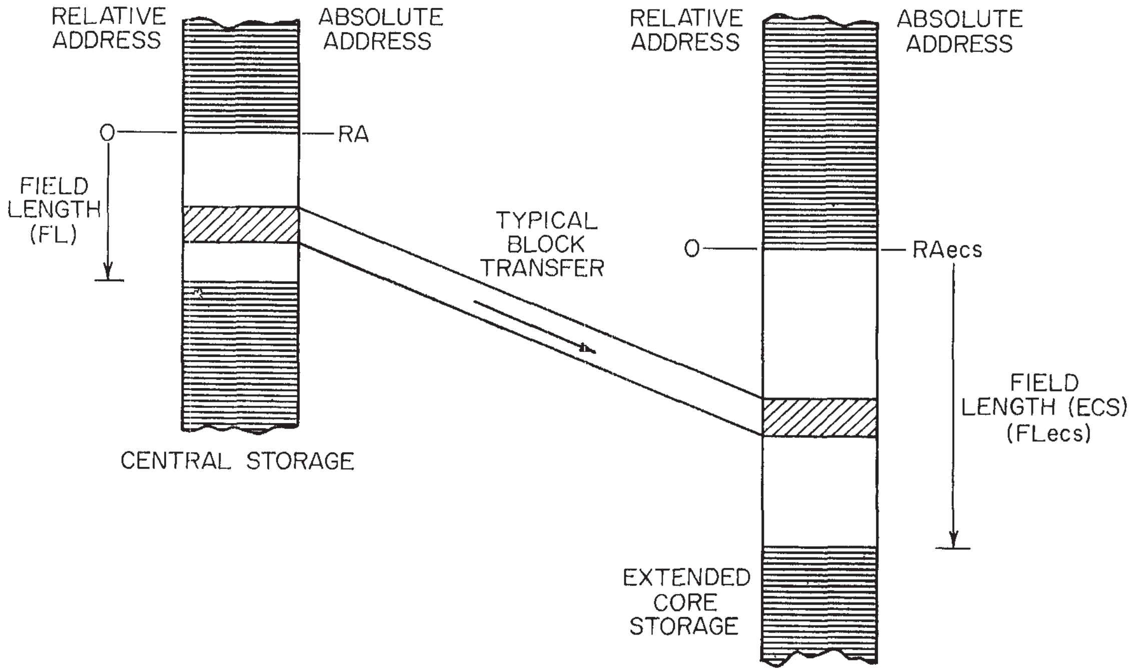

RELATIVE ADDRESS
ABSOLUTE ADDRESS
0
RA
FIELD LENGTH (FL)
CENTRAL STORAGE
TYPICAL BLOCK TRANSFER
RELATIVE ADDRESS
ABSOLUTE ADDRESS
0
RAecs
FIELD LENGTH (ECS) (FLecs)
EXTENDED CORE STORAGE

FIGURE 39

The two relative addresses are added by hardware at the time of a reference. A test of the address against the appropriate field length is also made at the time of a reference. No out-of-bounds references are allowed.

The Extended Core Storage is a linear-select 2-D magnetic core unit. This is a very large unit in terms of the number of bits in one bank. Although the read-write cycle time is over three times that of the central storage, the longer word length more than offsets, at least for block transfers. The block diagram of Figure 40 shows the ECS unit as two dimensional, with the word dimension of 488 bits, including 8 parity, and number of words equaling 15,744.

For purposes of separate identification, the 480-bit word is called a super-word, or sword. Note that the sword is disassembled for transfers to the 60-bit word used in Central Storage. For this operation the address received from the controller contains three lower order bits to identify one of the 60-bit words.

A typical block transfer can begin and end at any 60-bit word. However, because the nature of the transfer is a variable length block which usually contains more than one word, it is unnecessary to deliver an address for each 60-bit word. It is, in fact, convenient to deliver addresses only for swords. A "passive" form of control at the Extended Core Storage unit is used to transfer blocks. The initial word address causes the unit to reference the correct sword and assembles (disassembles) starting at the correct word within the sword. The assembly (disassembly) proceeds to the end of that sword in ascending word addresses. The next word address delivered to the unit will specify the first word in the next consecutive sword. This process continues until the last sword is referenced. This last reference includes assembly (disassembly) only to the last word specified.

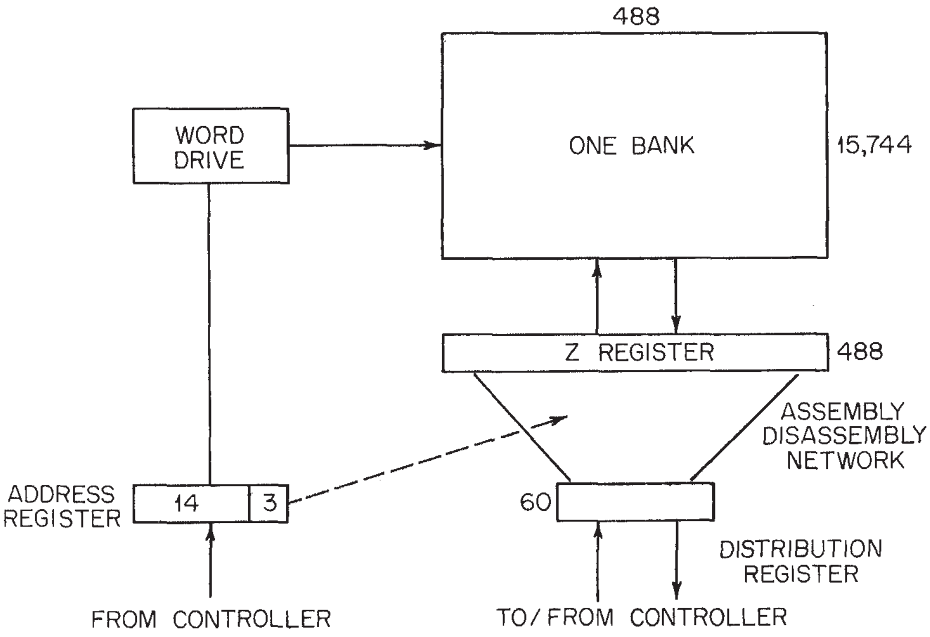


FIGURE 40

Banks of ECS may be interleaved in a manner similar to Central Storage. A typical block transfer is diagrammed in Figure 41, showing the interleaving and the beginning and ending cases.

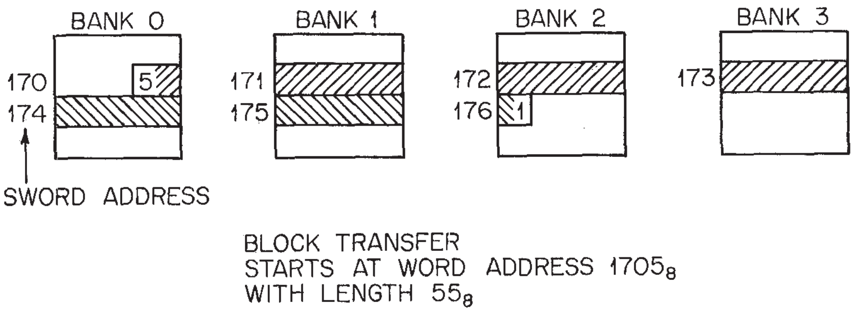

BANK 0
BANK 1
BANK 2
BANK 3
170
174
5
171
175
172
176
1
173
SWORD ADDRESS
BLOCK TRANSFER
STARTS AT WORD ADDRESS 1705₈
WITH LENGTH 55₈

FIGURE 41

With four interleaved banks it is possible to achieve a time overlap. For a transfer rate equal to the maximum rate of Central Storage, one 60-bit word must transfer every minor cycle, or 100 nanoseconds. Each ECS unit requires a 3200-nanosecond cycle, moving eight 60-bit words. Therefore, four banks can be made to move thirty-two words in the time of one storage cycle. To accomplish this, one bank is initiated every 800 nanoseconds.

F. ECS COUPLER AND CONTROLLER

To accomplish the connection of Extended Core Storage to Central Storage, two units are needed. The ECS Controller allows up to four 6600 Computers to be connected. Each 6600 Computer requires an ECS Coupler. See Figure 42.

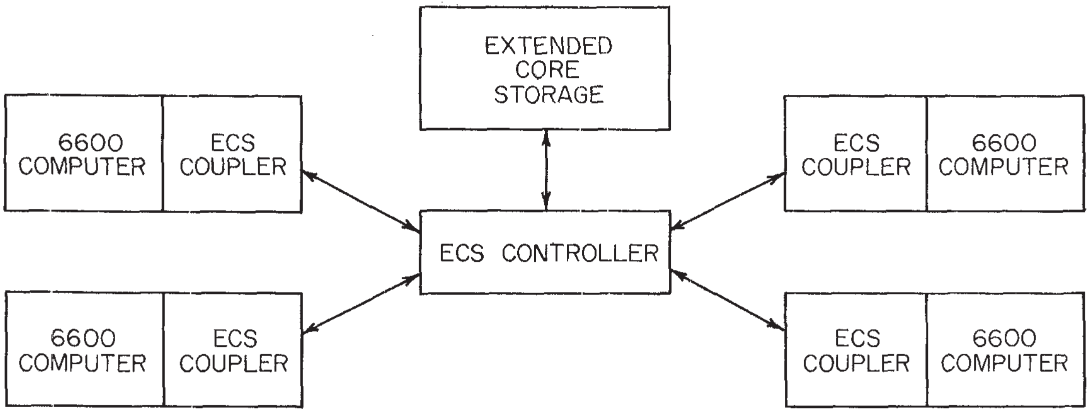

EXTENDED CORE STORAGE
6600 COMPUTER
ECS COUPLER
ECS COUPLER
6600 COMPUTER
ECS CONTROLLER
6600 COMPUTER
ECS COUPLER
ECS COUPLER
6600 COMPUTER

FIGURE 42

The ECS Coupler can be thought of as a functional unit of the 6600 CPU. However, no attempt is made to allow concurrent operation between it and other functions. In fact, any continuous transfer requires complete control over Central Storage. The CPU provides three items of data to perform the transfer.

- Initial Central Storage Address
- Initial Extended Core Storage Address
- Length of Transfer

From this initial data the ECS Coupler forms a continuous stream of Central Storage addresses and Extended Core Storage addresses, subject only to conflicts from other accesses to ECS and to PPU accesses to Central Storage. The ECS coupler allows such interruptions to occur at ECS sword boundaries. This is convenient for multiple access to ECS, each access channel effectively getting a sword at a time. In the case of an Exchange Jump Interrupt, the transfer is aborted at the next sword address.

The ECS Coupler includes counters which increment Central Storage and ECS addresses and the block length to zero. At the beginning of a block transfer, the block length is tested against the ECS bounds protection FL ecs. If this boundary is exceeded, the operation is aborted. Also, at the beginning the relative address RA ecs is added to the ECS address from the CPU to form the absolute initial address. From then on, the absolute address is used by the Coupler for ECS. The Central Storage relative address and bounds test is, of course, taken care of by the Stunt Box. Therefore, the ECS Coupler delivers addresses to the Stunt Box in the same relative manner as the CPU.

The ECS Controller provides access to four 6600 Computers. Each access channel is serviced on a round-robin basis. A scanner in the Controller tests requests each 800 nanoseconds to determine the next "user." If only one channel is active, the scanner is able to keep the total transfer rate at maximum. If, however, more than one channel is active, the rate per channel is reduced accordingly. This reduction is affected by the kind of bank conflict introduced.

CENTRAL PROCESSOR FUNCTIONAL UNITS

V

The central premise of the design of the Control Data 6600 is called "functional parallelism." Particularly in the Central Processor (CPU), this idea is essential to the design.

In special purpose digital devices hardware can be included to perform the necessary "housekeeping" tasks. By the use of this hardware, such tasks are eliminated from the apparent time to complete execution, since they are performed in parallel with the main functions being executed. The case is usually very specialized, however. This valuable technique can be applied to general purpose computers by including several independent functional units. Some of the units may be regarded as "housekeeping" units; others are the main processing units.

The 6600 Computer depends on the existence of separate functional units in the CPU for this kind of facility. Special purpose instructions are unnecessary to a great extent in this type of organization. Specialized handling of the control over the separate functions can, in theory, separate the critical path functions from the extras. In later sections examples will be given of the detailed effect of this type of control. In any event, the fact remains that some independence of function is necessary.

Ten functional units make up the arithmetic portion of the 6600 Central Processor. These are:

- Boolean Unit
- Shift Unit

- Fixed Add
- Floating Add
- Multiply (2)
- Divide
- Increment (2)
- Branch

Two identical units are provided for the Multiply and Increment functions for emphasis. The logical properties of these units are given in the block diagram of Figure 43.

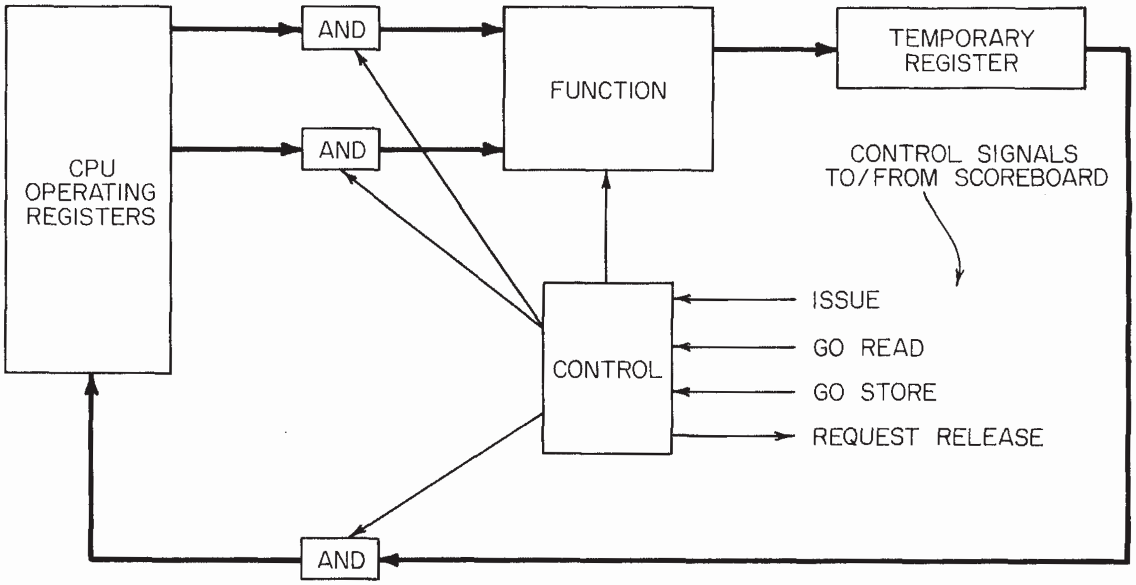


FIGURE 43

The operation is generally three address in nature, with some exceptions. When both operands are available in the CPU Registers, the input operands are transferred to the functional unit. On completion, the result is temporarily deposited in a result register. The control system is responsible for removing this result, transferring it to the CPU Registers, and releasing the functional unit.

Because the functional units are essentially separate and independent, the design of each unit can be quite specific and optimum. The selection of functions and the properties of each function are consistent with this independence. Total time for a function includes the "extra" minor cycle to send a result to the CPU registers and through to a point on the register output bus.

Instruction formats within the Central Processor are 15 bit and 30 bit as shown on page 59.

The 15-bit format is the principal one used, which is testimony to the value of the CPU registers. While past computers utilized a single "accumulator" register to good advantage, the instruction required a storage address similar to the K field shown. This caused inefficiencies in two ways, if not

F	m	i	j	k
3	3	3	3	3

15 bit

F	m	i	j	K
3	3	3	3	18

30 bit

Where F—denotes the major class of function,
m—denotes a mode within the functional unit,
i—identifies one of eight registers within the proper group of X, B, or A registers,
j—identifies one of eight registers within the proper group,
k—identifies one of eight registers within the proper group,
K—an 18-bit immediate field used as a constant or Branch destination.

more. A first, and obvious, inefficiency was the length of the instruction word itself, using storage space and time to fetch. A second inefficiency was the use of single-address procedures, implying the single accumulator register. Going beyond single-address, of course, makes the instruction length even worse.

With the introduction of additional CPU registers, the problems of instruction length and of single-address technique are removed. The 6600 Central Processor can utilize a very efficient three-address scheme, as shown, with very short instructions.

Although each register designator, i, j, and k, refers to one of eight registers; the function F controls the proper class of register. Three classes are included, as follows:

X	60-bit Operand Registers	8 used.
B	18-bit Index Registers	8 used.
A	18-bit Address Registers	8 used.

The X registers are the principal transient registers for data words within the Central Processor. Binary fixed-point numbers, floating-point numbers, packed alphanumeric data, and so on are handled through the X registers. In general, data words enter a specific X register from Central Storage; they are operated on by functional units and are finally returned to Central Storage from a specific X register.

The 18-bit Address registers control the Central Storage references and can be indexed by appropriate use of the 18-bit Index registers. The Index registers are also convenient for fixed-point integers, floating-point exponent manipulation, control of shifts, and so on.

A. BOOLEAN UNIT

Of all the functional units, the Boolean unit is the simplest and easiest to describe. This is, of course, the logical unit, performing the fundamental logical operations defined in Boolean algebra. Eight functions are provided as described in the list of instructions below. Note that the total instruction list is provided in Appendix A.

10	Transfer Xj to Xi.
11	Logical Product of Xj and Xk to Xi.
12	Logical Sum of Xj and Xk to Xi.
13	Logical Difference of Xj and Xk to Xi.
14	Transfer Xk complement to Xi.
15	Logical Product of Xj and Xk complement to Xi.
16	Logical Sum of Xj and Xk complement to Xi.
17	Logical Difference of Xj and Xk complement to Xi.

These instructions describe the logical functions operating on input operands Xj and Xk with the result going to Xi. These operands and the result are held in the CPU Registers of 60-bit word length, identified as the X Registers. The subscript identification i, j, or k refers to a designator field in the instruction referring to the correct register.

The terms "Logical Sum," "Logical Product," and "Logical Difference" refer to the following.

Logical Sum	Inclusive OR
Logical Product	AND
Logical Difference	Exclusive OR

The Boolean Unit requires 300 nanoseconds, or three minor cycles from the time input operands are sampled, until the result is available in the CPU Registers for sampling by another functional unit. This is a minimum case and may be longer if the result is held up because of other use of the result register.

A detailed block diagram is shown in Figure 44.

Operands are entered in input registers, the data coming from Xj and Xk registers. Five control bits are stored in the Boolean unit at the time of ISSUE, essentially preparing it for the desired operation. The Scoreboard unit determines when the operands are sent to the unit and sends a GO BOOLEAN signal. This signal enters the operands into the unit, performing a complement as necessary. The complement is the Boolean NOT and is only required on the Xk entry path.

Once the data operands are entered, a period of time dependent on the "distance" through the network is required before the result can be stored. This time is shown as 125 nanoseconds in Figure 44. The "request release" signal is sent to the SCOREBOARD at a time sufficiently early in the unit's operation to allow a GO TRANSMIT signal to return just after the result has been stored in the temporary result register. This timing is such that the request is sent *before* the input operands arrive at the Boolean unit!

A detailed logic diagram for one bit is shown in Figure 45. The Boolean unit is mounted in one chassis (number two) and utilizes four types of logic module, excluding control. Data enters an input register from the coaxial

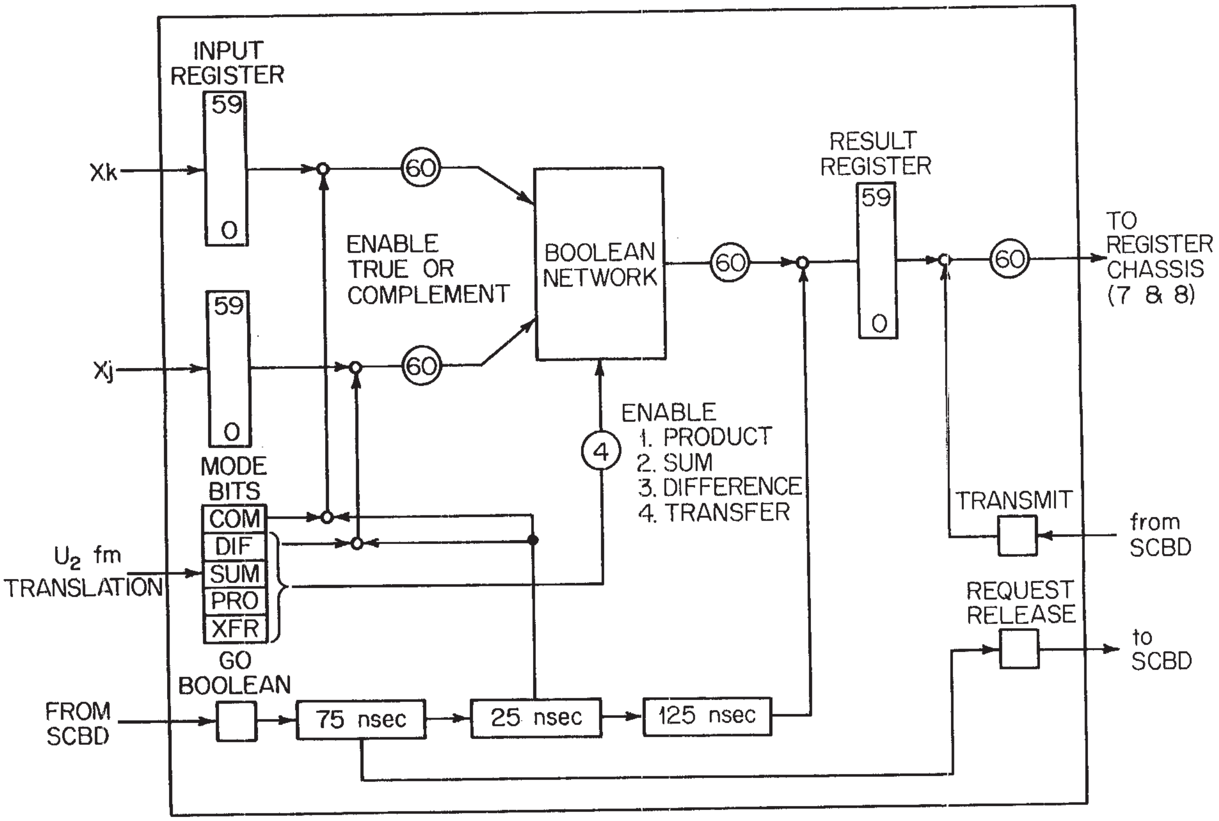

INPUT REGISTER
59
0
Xk
Xj
60
ENABLE TRUE OR COMPLEMENT
BOOLEAN NETWORK
RESULT REGISTER
TO REGISTER CHASSIS (7 & 8)
ENABLE
1. PRODUCT
2. SUM
3. DIFFERENCE
4. TRANSFER
4
MODE BITS
COM
DIF
SUM
PRO
XFR
U₂ fm TRANSLATION
TRANSMIT
from SCBD
REQUEST RELEASE
to SCBD
GO BOOLEAN
FROM SCBD
75 nsec
25 nsec
125 nsec

FIGURE 44 The Boolean functional unit.

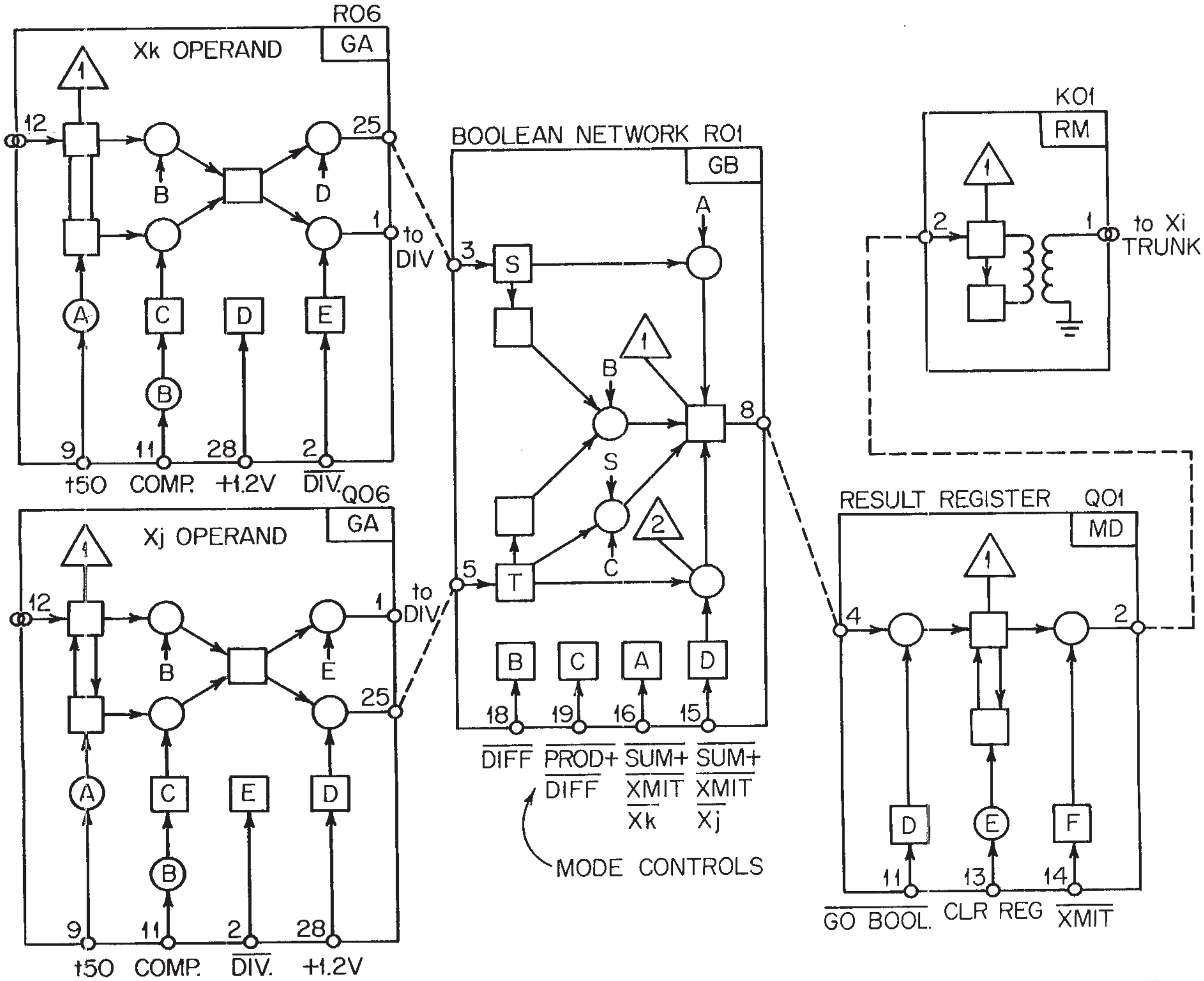

R06
GA
Xk OPERAND
12
25
1
to DIV
B
D
A
C
D
E
9
11
28
2
+50
COMP.
+1.2V
DIV
Q06
GA
Xj OPERAND
12
1
to DIV
25
B
E
A
C
E
D
9
11
2
28
+50
COMP.
DIV.
+1.2V
BOOLEAN NETWORK R01
GB
3
5
S
T
A
B
S
C
8
1
2
B
C
A
D
18
19
16
15
DIFF
PROD+ DIFF
SUM+ XMIT Xk
SUM+ XMIT Xj
MODE CONTROLS
K01
RM
2
1
to Xi TRUNK
RESULT REGISTER Q01
MD
4
2
D
E
F
11
13
14
GO BOOL.
CLR REG
XMIT

FIGURE 45

cable in the form of a 25-nanosecond pulse. Mode bits have been established in advance of this time. Therefore, the network following these input registers will react to the data entering. This forms a stable result as an output of the GB module by the time delayed GO BOOLEAN samples the result into the result register.

The control signals are steady signals with the exception of the GO BOOLEAN, TRANSMIT, and the CLEAR REGISTER signals. The complement signal enters the GA modules, forming the control terms B and C. B selects the true output, and C selects the complement output; the result is OR'ed and sent to the Boolean Network Module GB. Four control signals are used to control the network module. These signals are shown in derived form in Table IV.

TABLE IV Boolean Functional Unit Reference Chart

Code	Name	Function	Mode Bits					GB Module						
			Sum	Prod	Xfer	Comp	Diff	P3	P5	A	B	C	D
10	TRANSMIT	$Xj \rightarrow Xi$			X			$\overline{Xj}$	$\overline{Xk}$				X
11	LOGICAL PRODUCT	$Xj \cdot Xk$		X				$\overline{Xj}$	$\overline{Xk}$			X	
12	LOGICAL SUM	$Xj + Xk$	X					$\overline{Xj}$	$\overline{Xk}$	X			X
13	LOGICAL DIFFERENCE	$Xj - Xk$					X	Xj	$\overline{Xk}$		X	X	
14	TRANSMIT	$\overline{Xk} \rightarrow Xi$			X	X		$\overline{Xj}$	Xk	X			
15	LOGICAL PRODUCT	$Xj \cdot \overline{Xk}$		X		X		$\overline{Xj}$	Xk			X	
16	LOGICAL SUM	$Xj + \overline{Xk}$	X			X		$\overline{Xj}$	Xk	X			X
17	LOGICAL DIFFERENCE	$Xj - \overline{Xk}$				X	X	Xj	Xk		X	X	

The network in module GB is an excellent example of the intimate design relationship of circuit ground rules and logic. The output of the GB module can be the AND of the inputs, the inclusive OR, the exclusive OR, or a selection of one input. Control is accomplished by the four terms A, B, C, or D. Combinations of these controls provide for these outputs. Control of the complement of input operand Xk is accomplished on GA.

The logic is split between the modules in order to gain an optimum use of pins, module types, and especially to gain the maximum load on control terms. For example, the GA module actually contains four bits of one operand, either Xj or Xk. Therefore, the loads on the control terms A, C, D, and E are four each, one less than the ground rule limit. However, the load on control term B is five, or the limit.

An additional limit is the number of transistors per module, a nominal maximum of 64. By counting arrowheads on a GA module in the diagram of Figure 45, it can be seen that fourteen transistors per bit plus five transistors for control are required, totaling 61. Pin count on this module is three per bit plus four control, totaling 16 and not even coming close to the maximum of 28.

With the same kind of constraint on the GB module, only three bits of network are contained on each. The transistor count of 18 per bit and four control totals 58.

B. FIXED ADD UNIT

The Fixed Add Unit is the next in ascending order of complexity. The unit is an integer arithmetic unit which performs one's-complement fixed-point addition or subtraction on 60-bit numbers. The one's-complement representation of binary numbers refers to the treatment of negative numbers. The sign bit is the left-most, or most significant bit of the number. If the sign bit is zero, the number is considered positive with its binary point just to the right of the least significant bit. If the sign bit is one, the number is considered negative with similar binary point placement and with each "zero" acting as a "one" does in the positive case. This can be charted as in Figure 46.

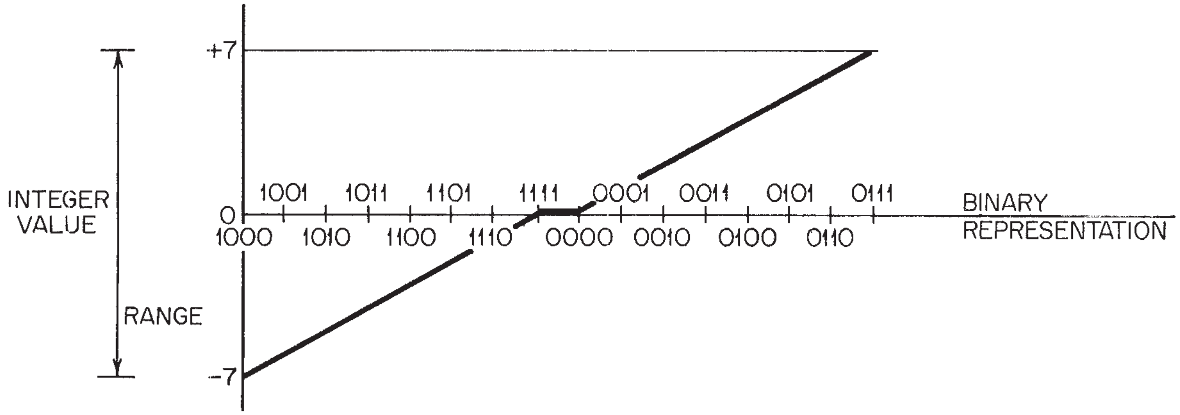


FIGURE 46

This chart is made for a register length of only four bits but shows the condition of a positive and negative zero representation.

The Fixed Add Unit uses a scheme of parallel addition which forms all bits of the sum simultaneously. To explain the technique, consider a six-bit add unit operating as a serial adder. One can construct a set of equations describing the step-by-step results of addition.

$$\mathrm{Sum\ n} = \mathrm{AnBn}\overline{\mathrm{C}}\mathrm{n} + \overline{\mathrm{A}}\mathrm{nBn}\overline{\mathrm{C}}\mathrm{n} + \overline{\mathrm{A}}\mathrm{n}\overline{\mathrm{B}}\mathrm{nCn} + \mathrm{AnBnCn},$$

where An = 1st operand bit n,
Bn = 2nd operand bit n,
Cn = Carry *entering* the nth bit.

Note that one's complement arithmetic requires an "end-around" carry from the most significant bit to the least significant. In bit serial adders this, therefore, requires another pass through the adder.

$$\mathrm{Carry\ n} + 1 = \mathrm{AnBn}\overline{\mathrm{C}}\mathrm{n} + \mathrm{An}\overline{\mathrm{B}}\mathrm{nCn} + \overline{\mathrm{A}}\mathrm{nBnCn} + \mathrm{AnBnCn}. \qquad \text{(V-2)}$$

Simplifying these two equations,

$$\mathrm{Sum\ n} = (\mathrm{An} \oplus \mathrm{Bn}) \oplus \mathrm{Cn}, \qquad \text{(V-3)}$$

where $\oplus$ is the symbol for Exclusive OR.

$$\mathrm{Carry\ n} + 1 = \mathrm{AnBn} + (\mathrm{An} + \mathrm{Bn})\mathrm{Cn}. \qquad \text{(V-4)}$$

Of interest is the case of the sum of +1 and −1, using the above equations.

$$\begin{array}{rcl} \mathrm{A} = 0\ 0\ 0 & 0\ 0\ 1 & +1 \\ \mathrm{B} = 1\ 1\ 1 & 1\ 1\ 0 & -1 \\ \mathrm{Carry\ n} + 1 = 0\ 0\ 0 & 0\ 0\ 0 & \\ \mathrm{Sum\ n} = 1\ 1\ 1 & 1\ 1\ 1 & \end{array}$$

This is, of course, the "negative" zero and is a perfectly correct answer. It makes, however, for difficulty in interpretation and in conversion to display code or print code. This negative zero arises from the choice above of an "additive" type of adder. To produce a "positive" zero result it is necessary to construct a "subtractive" type of adder. To show this type a similar set of equations is given below.

$$\mathrm{Sum\ n} = \mathrm{XnYn}\overline{\mathrm{B}}\mathrm{n} + \mathrm{Xn}\overline{\mathrm{Y}}\mathrm{nBn} + \overline{\mathrm{X}}\mathrm{nYnBn} + \overline{\mathrm{X}}\mathrm{n}\overline{\mathrm{Y}}\mathrm{n}\overline{\mathrm{B}}\mathrm{n}, \qquad \text{(V-5)}$$

where Xn = 1st operand bit n
Yn = 2nd operand bit n
Bn = Borrow *entering* bit n

$$\mathrm{Borrow\ n} + 1 = \overline{\mathrm{X}}\mathrm{n}\overline{\mathrm{Y}}\mathrm{nBn} + \mathrm{Xn}\overline{\mathrm{Y}}\mathrm{n}\overline{\mathrm{B}}\mathrm{n} + \overline{\mathrm{X}}\mathrm{nYn}\overline{\mathrm{B}}\mathrm{n} + \overline{\mathrm{X}}\mathrm{n}\overline{\mathrm{Y}}\mathrm{n}\overline{\mathrm{B}}\mathrm{n} \qquad \text{(V-6)}$$

Simplifying these equations,

$$\mathrm{Sum\ n} = (\mathrm{X} \oplus \mathrm{Y}) \oplus \overline{\mathrm{B}}\mathrm{n} \qquad \text{(V-7)}$$

$$\mathrm{Borrow\ n} + 1 = \overline{\mathrm{X}}\mathrm{n}\overline{\mathrm{Y}}\mathrm{n} + (\overline{\mathrm{X}}\mathrm{n} + \overline{\mathrm{Y}}\mathrm{n})\mathrm{Bn} \qquad \text{(V-8)}$$

Now, taking the same case as above,

$$\begin{array}{llll}
X = 0\ 0\ 0 & 0\ 0\ 1 & & +1 \\
Y = 1\ 1\ 1 & 1\ 1\ 0 & & -1 \\
\\
\text{Borrow } n + 1 = & 0\ 0\ 0 & 0\ 0\ 0 & \\
\text{Sum} = & 0\ 0\ 0 & 0\ 0\ 0 &
\end{array}$$

The result is a more natural and convenient form. Note that only one combination of operands to such a "subtractive" adder will produce a negative zero, that is, two negative zeros as input. This complication of one's complement arithmetic is belabored here because of the need to test for the zero case simply.

The above exercise is given as a preparation for the discussion of the parallel network used in the Fixed Add unit. A parallel network can be made to act in the same manner as indicated in the above exercise. Again taking a six-bit case, the following series can be formed.

$$\text{Sum } n = (Xn \oplus Yn) \oplus \bar{B}n \qquad \text{(V-9)}$$

$$\text{Borrow } n + 1 = \bar{X}n\bar{Y}n + (\bar{X}n + \bar{Y}n)Bn \qquad \text{(V-10)}$$

Forming Borrows in groups of three,

$$B_1 = \bar{X}_0\bar{Y}_0 + (\bar{X}_0 + \bar{Y}_0)B_0$$

$$\begin{aligned} B_2 &= \bar{X}_1\bar{Y}_1 + (\bar{X}_1 + \bar{Y}_1)B_1 \\ &= \bar{X}_1\bar{Y}_1 + \bar{X}_0\bar{Y}_0(\bar{X}_1 + \bar{Y}_1) + (\bar{X}_0 + \bar{Y}_0)(\bar{X}_1 + \bar{Y}_1)B_0 \end{aligned}$$

$$\begin{aligned} B_3 &= \bar{X}_2\bar{Y}_2 + (\bar{X}_2 + Y_2)B_2 \\ &= \bar{X}_2\bar{Y}_2 + \bar{X}_1\bar{Y}_1(\bar{X}_2 + \bar{Y}_2) + \bar{X}_0\bar{Y}_0(\bar{X}_1 + \bar{Y}_1)(\bar{X}_2 + \bar{Y}_2) \\ &\qquad + (\bar{X}_0 + \bar{Y}_0)(\bar{X}_1 + \bar{Y}_1)(\bar{X}_2 + \bar{Y}_2)B_0 \end{aligned}$$

$$B_4 = \bar{X}_3\bar{Y}_3 + (\bar{X}_3 + \bar{Y}_3)B_3$$

$$\begin{aligned} B_5 &= \bar{X}_4\bar{Y}_4 + (\bar{X}_4 + \bar{Y}_4)B_4 \\ &= \bar{X}_4\bar{Y}_4 + \bar{X}_3\bar{Y}_3(\bar{X}_4 + \bar{Y}_4) + (\bar{X}_3 + \bar{Y}_3)(\bar{X}_4 + \bar{Y}_4)B_3 \end{aligned}$$

$$\begin{aligned} B_6 &= \bar{X}_5\bar{Y}_5 + (\bar{X}_5 + \bar{Y}_5)B_5 \\ &= \bar{X}_5\bar{Y}_5 + \bar{X}_4\bar{Y}_4(\bar{X}_5 + \bar{Y}_5) + \bar{X}_3\bar{Y}_3(\bar{X}_4 + \bar{Y}_4)(\bar{X}_5 + \bar{Y}_5) \\ &\qquad + (\bar{X}_3 + \bar{Y}_3)(\bar{X}_4 + \bar{Y}_4)(\bar{X}_5 + \bar{Y}_5)B_3 \end{aligned}$$

To close this six-bit adder for one's-complement use, B_6 is made equal to B_0. Examination of the logic of the add network derived above will show that no insoluble cases exist.

For the purpose of clarity, the logical terms above on the right-hand sides which do not include a B term are defined as Borrow Generation terms. The logical terms which AND with a B term are called Borrow Pass terms.

For example,

$$B_5 = \underbrace{\overline{X}_4\overline{Y}_4 + \overline{X}_3\overline{Y}_3(\overline{X}_4 + \overline{Y}_4)}_{\text{Borrow Generator}} + \underbrace{(\overline{X}_3 + \overline{Y}_3)(\overline{X}_4 + \overline{Y}_4)B_4}_{\text{Borrow Pass}}$$

It should be evident that *all* Borrow Generate and Borrow Pass terms can be simultaneously solved. However, the Borrow terms themselves must wait for these. For the description of this portion of the Fixed Add Unit with some detail, see Figure 47.

This drawing simplifies the discussion by showing the "interface" information passing from module to module within the Fixed Add Unit. Although only six bits were defined, it is a straightforward exercise to develop the completed network.

Operands X and Y refer to the two input operands to the unit. Groups of three bits each are entered into each FA module.

1. $X_0Y_0 + \overline{X}_0\overline{Y}_0$
2. X_0Y_0 (Inverted)
3. $\overline{X}_0\overline{Y}_0$ (Inverted)
4. $X_1Y_1 + \overline{X}_1\overline{Y}_1$
5. $\overline{X}_1\overline{Y}_1$ (Inverted)
6. $X_2Y_2 + \overline{X}_2\overline{Y}_2$
7. $(\overline{X}_0 + \overline{Y}_0)(\overline{X}_1 + \overline{Y}_1)(\overline{X}_2 + \overline{Y}_2)$
8. $\overline{X}_2\overline{Y}_2 + \overline{X}_1\overline{Y}_1(\overline{X}_2 + \overline{Y}_2) + \overline{X}_0\overline{Y}_0(\overline{X}_1 + \overline{Y}_1)(\overline{X}_2 + \overline{Y}_2)$

The first six of these are wired directly to a "companion" FE module to complete the local portion of the network. Item 7 is called the Borrow Pass for the three-bit group. Item 8 is correspondingly called the Borrow Generation from this three-bit group.

Four groups of three-bit combinations are handled within the FB module. Similarly, five sections of such groups are combined in the FD module to complete the borrow logic for all 60 bits. Summarizing the usage of these modules:

Module FA—20 used—contains three bits each of input operands X and Y;
Module FB—5 used—combines intermediate borrow logic for each section of twelve bits;
Module FC—1 used—combines intermediate borrow pass terms;
Module FD—4 used—combines borrow logic for the sections;
Module FE—20 used—completes the sum and borrow logic to form the result for three bits each.

This add network illustrates the case of multiple paths through a network with different "lengths." For example, the longest path in Figure 47 requires sixteen inversions, not including the input flip-flop. Similarly, the shortest path is only five inversions, again not including the input flip-flop.

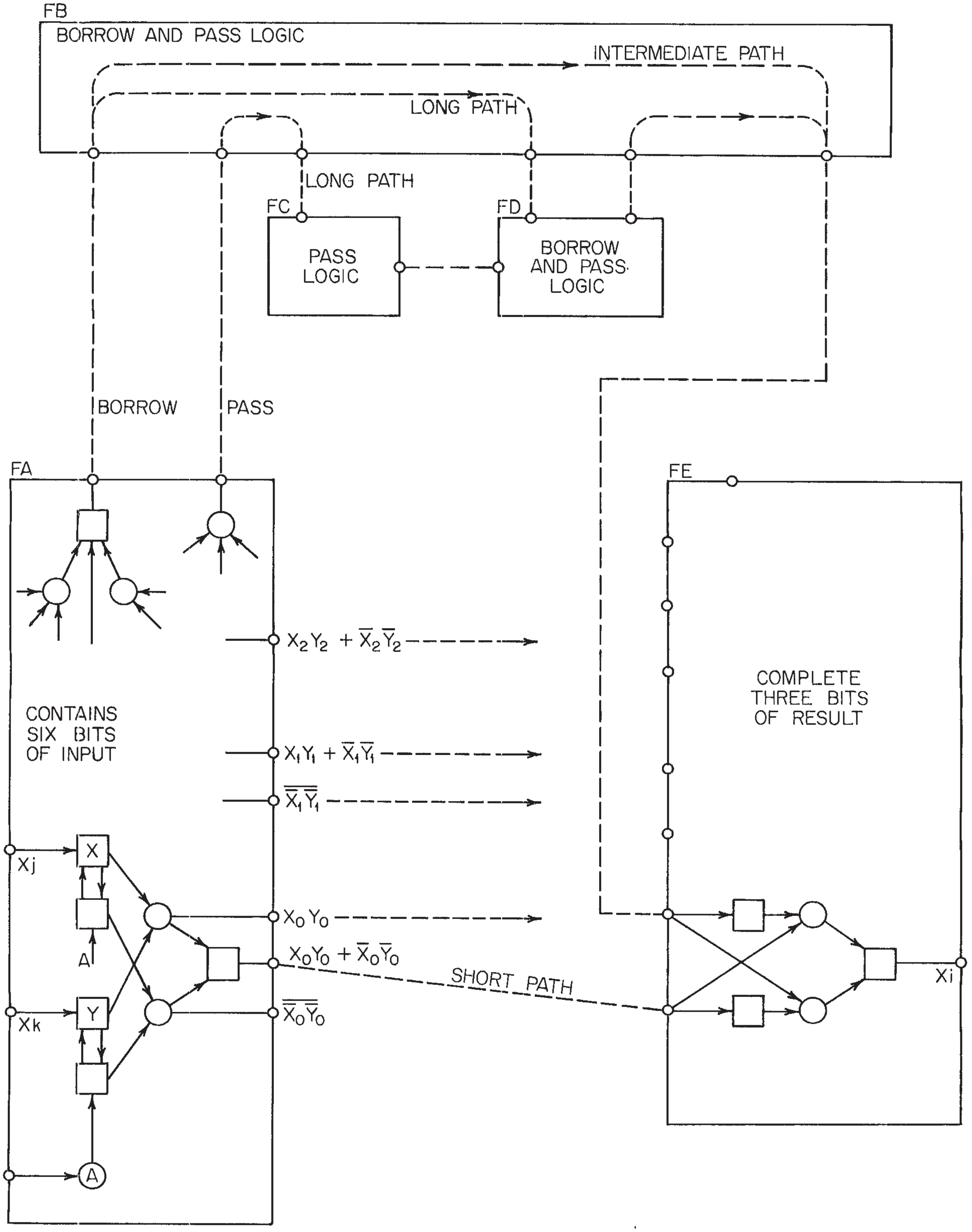

FB
BORROW AND PASS LOGIC
INTERMEDIATE PATH
LONG PATH
LONG PATH
FC
FD
PASS LOGIC
BORROW AND PASS LOGIC
BORROW
PASS
FA
FE
$X_2Y_2 + \overline{X_2}\overline{Y_2}$
CONTAINS SIX BITS OF INPUT
$X_1Y_1 + \overline{X_1}\overline{Y_1}$
$\overline{\overline{X_1}\overline{Y_1}}$
COMPLETE THREE BITS OF RESULT
X
Xj
A
Y
Xk
A
X_0Y_0
$X_0Y_0 + \overline{X_0}\overline{Y_0}$
SHORT PATH
$\overline{\overline{X_0}\overline{Y_0}}$
Xi

FIGURE 47

A number of intermediate lengths can be seen in the network also. In the case of the Fixed Add Unit, the differing path lengths cause no difficulty because the input registers are held stable until all paths are stable.

The block diagram of the Fixed Add Unit is given in Figure 48. In this unit, no temporary result register is needed because the input registers serve

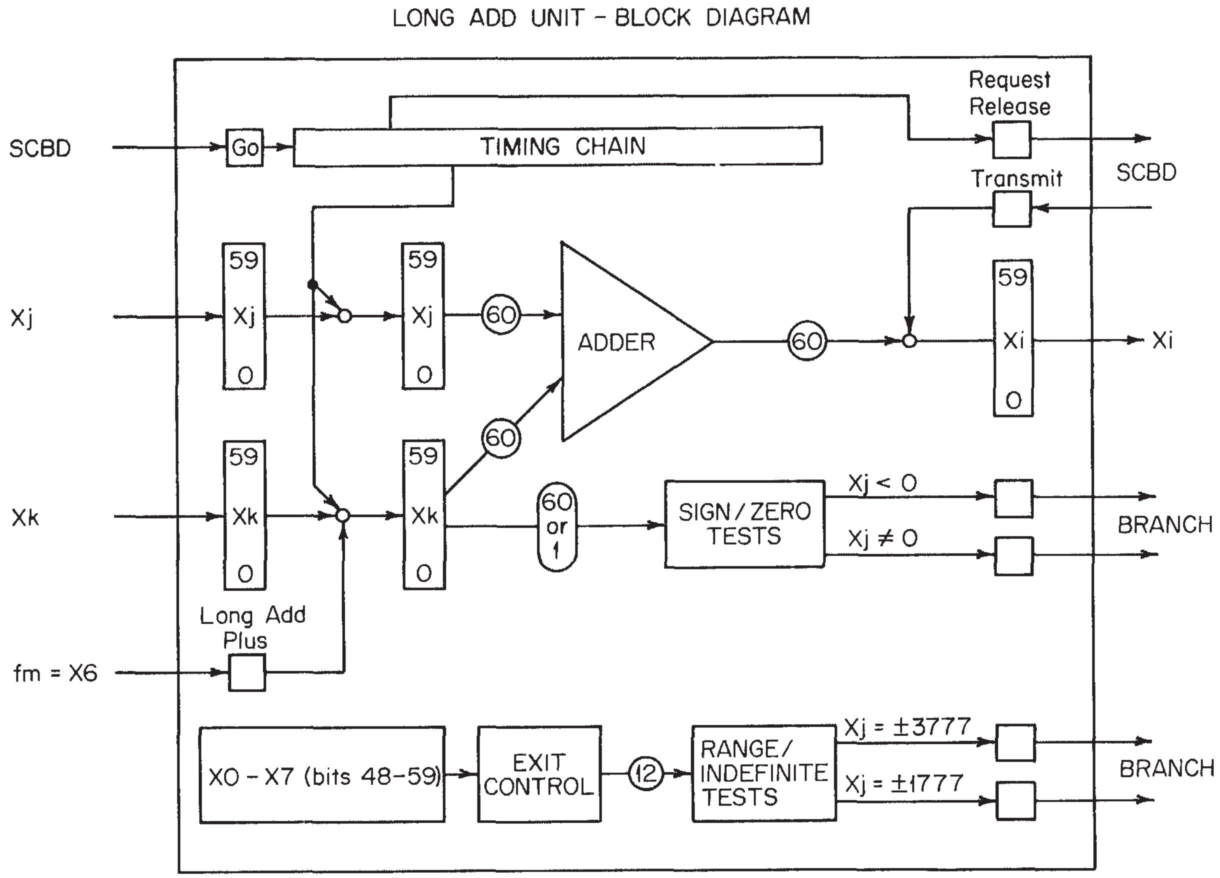

LONG ADD UNIT - BLOCK DIAGRAM
SCBD
Go
TIMING CHAIN
Request
Release
Transmit
SCBD
Xj
59
Xj
0
60
ADDER
60
59
Xi
0
Xi
Xk
Xk
60
60
or
1
SIGN/ZERO
TESTS
Xj < 0
Xj ≠ 0
BRANCH
Long Add
Plus
fm = X6
X0 - X7 (bits 48-59)
EXIT
CONTROL
12
RANGE/
INDEFINITE
TESTS
Xj = ±3777
Xj = ±1777
BRANCH

FIGURE 48 Long add unit—block diagram.

no other function and can be held stable until the result is sent direct to the CPU register.

Only two CPU instructions are executed by the Fixed Add Unit.

36	Integer Sum of Xj and Xk to Xi.
37	Integer Difference of Xj and Xk to Xi.

These two operations are performed on 60-bit quantities. The Integer Difference is accomplished by loading the complement of register Xk into one of the Unit input registers.

The Fixed Add Unit is also utilized as a "partner" to the Branch unit for conditional tests of selected X registers, as follows.

Conditional Branch Instructions on Xj

030	Go to K if Xj = 0
031	Go to K if Xj ≠ 0
032	Go to K if Xj is positive
033	Go to K if Xj is negative
034	Go to K if Xj is in range
035	Go to K if Xj is out of range
036	Go to K if Xj is definite
037	Go to K if Xj is indefinite

These instructions will be described in more detail in Chapter VI. For the purpose of this discussion, only the usage of the Fixed Add Unit is of interest. Tests are made in the following manner.

030 and 031—The zero tests check the full 60-bit word in Xj. Both "positive" and "negative" zero are considered zero in this test, thereby allowing use for fixed and floating point numbers.

032 and 033—The sign tests check only the most significant bit of Xj, the sign bit. Tests are valid for both fixed and floating point numbers.

034 and 035—The range tests check the most significant twelve bits of Xj for the floating point represention of infinity. This is defined as 3777 (octal) for positive infinity and 4000 for negative infinity, with all lower order bits ignored. The number in Xj is assumed to be a floating point number.

036 and 037—The definite/indefinite tests check the most significant twelve bits of Xj for the floating point representation of indefinite. This is defined as 1777 for positive indefinite and 6000 for negative indefinite, with all lower order bits ignored. The number in Xj is assumed to be a floating point number.

The above tests are made during the execution of the corresponding Branch instruction. At the start of the Branch, *both* the Branch Unit and the Fixed Add Unit are started together, as "partners." The Fixed Add Unit proceeds through its execution sequence providing only the sign tests and zero tests. The range and indefinite tests are performed external to the unit on its input bus coming from the CPU X registers, as a convenience.

C. DATA TRUNKS

In the preceding discussion of the Boolean and Fixed Add Units, it was apparent that one-third of the execution time is taken up in transferring data to and from the CPU registers. A considerable percentage of the CPU hardware is devoted to the data trunks for this task. To balance this hardware against effective performance of the functional units, several groupings are made as shown in Figure 49 (page 70).

In the case of the Boolean and Fixed Add Units, there are two 60-bit input trunks and one 60-bit result trunk. Three independent trunks are involved insofar as the CPU registers are concerned. These trunks are designed to allow simultaneous traffic on all four sets of trunks as shown in Figure 49. This means that two input operands may transfer to a selected functional unit in each of Groups I, II, and III. Also, one 60-bit word may transfer to Central Storage for a total simultaneous readout of seven registers in any one minor cycle. The trunks are designed to be reused on a new selection every minor cycle.

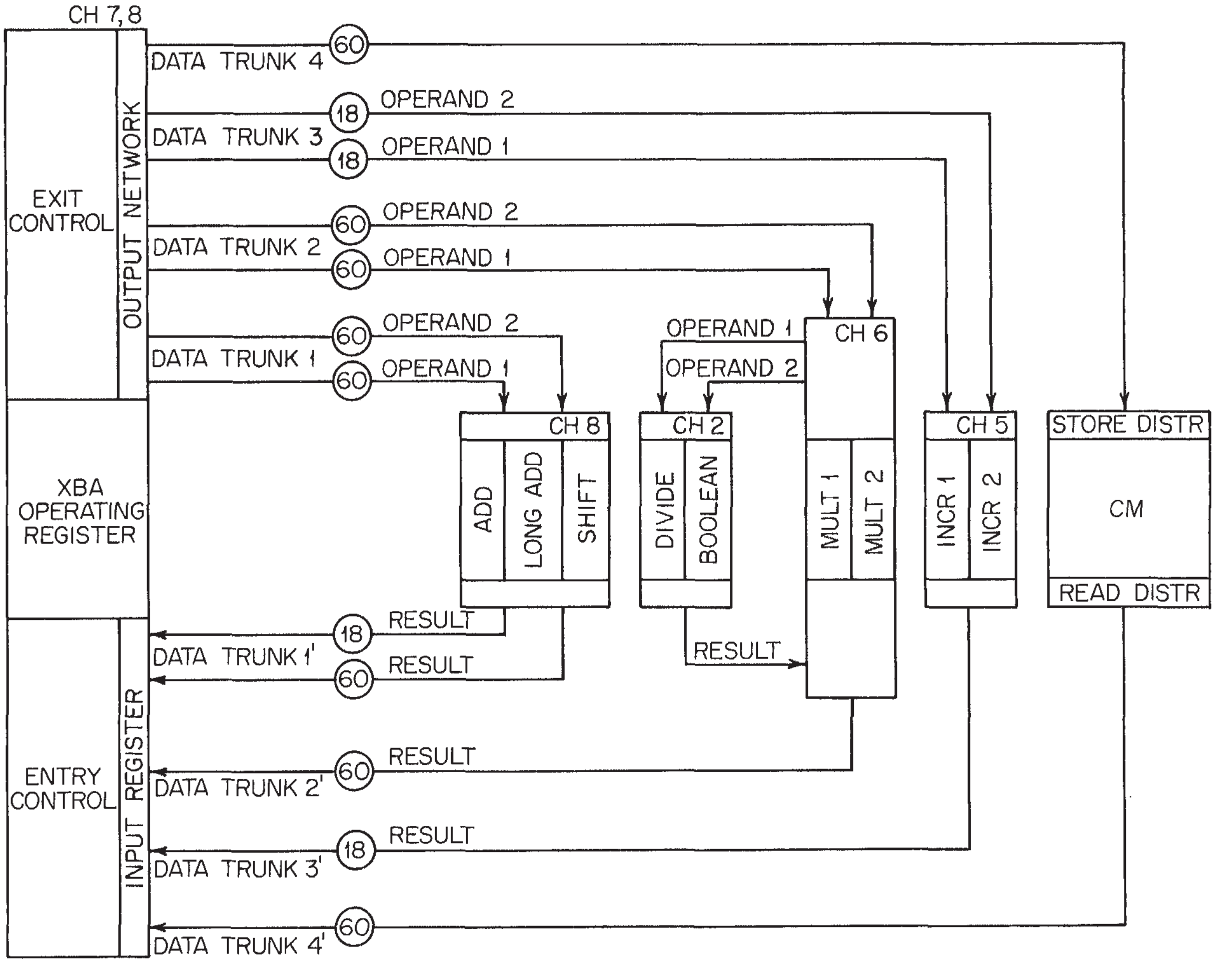

CH 7,8
EXIT CONTROL
OUTPUT NETWORK
XBA OPERATING REGISTER
ENTRY CONTROL
INPUT REGISTER
DATA TRUNK 4
DATA TRUNK 3
DATA TRUNK 2
DATA TRUNK 1
OPERAND 1
OPERAND 2
60
18
CH 8
ADD
LONG ADD
SHIFT
CH 2
DIVIDE
BOOLEAN
CH 6
MULT 1
MULT 2
CH 5
INCR 1
INCR 2
STORE DISTR
CM
READ DISTR
RESULT
DATA TRUNK 1'
DATA TRUNK 2'
DATA TRUNK 3'
DATA TRUNK 4'

FIGURE 49 Data trunks.

Going in the reverse direction, only one result is transferred from a functional unit to the CPU registers, with the special exception of the Shift Unit. Again each grouping can transfer a word simultaneously for a total input to the CPU registers of five words in any one minor cycle.

The groupings of functional units are chosen for physical placement reasons as well as certain performance advantages. For example, it is an advantage to the ADD unit to be located in Chassis 8 in close proximity to the most significant bits of the X register. This convenience, however, limited the available space for other functional units. Note also that the Boolean and Divide Units are connected to the data trunks through an intermediate chassis on which are contained the Multiply Units.

A further choice in the groupings relates to the level of data trunk traffic expected. As will be seen, the two Increment Units can cause a traffic of one operand on a trunk every two minor cycles. The group of Multiply Units and Divide Unit, excluding Boolean, can cause a traffic of one operand on a trunk in just over every four minor cycles. With the Boolean Unit able to cause one operand on a trunk every four minor cycles, this grouping nearly matches the Increment Unit grouping.

The data trunks are so arranged that a result can be entered into a selected register and be immediately transferred on a trunk to the input of a functional unit, all within one minor cycle. For the purpose of describing the time taken by a functional unit, this minor cycle is always included. For example, the Boolean time is two minor cycles, starting from the input operands, plus one minor cycle "through" the result register, for a total of three minor cycles. Obviously, any conflict in the use of a data trunk by members of a group will add a minor cycle to the "loser." To resolve conflicts, a fixed priority system is used in each group. These are listed in the order of descending priority within each group below.

	Read Operand Trunk	Result Trunk
Group I	Divide	Boolean
	Multiply I	Divide
	Multiply II	Multiply I
	Boolean	Multiply II
Group II	Add	Shift
	Shift	Add
	Fixed Add	Fixed Add
Group III	Increment I	Increment I
	Increment II	Increment II

The priorities are slightly different for the Read Operand Trunk and the Result Trunk in deference to the level of "traffic" expected on each.

Additional discussion of the exit and entry control of the CPU registers in Chapter VI will further clarify the extent of the hardware in the data trunks.

D. SHIFT UNIT

The Shift Unit performs shift, normalize, pack, unpack, and mask operations. The execution time of normalize operations is 400 nanoseconds, or one minor cycle greater than the other Shift operations. Again, this time includes a minor cycle to store the results in the CPU registers.

The Shift Unit is slightly different from other functional units. For example, the jk field of all Shift instructions is inserted in the unit directly from the instruction at ISSUE time, for use in certain instructions. Also, the Shift Unit has two result trunks to handle the normalize and unpack instructions.

The following instructions are listed for the Shift Unit.

20 Shift Xi Left jk places
21 Shift Xi Right jk places

22 Shift Xk nominally Left Bj places to Xi
23 Shift Xk nominally Right Bj places to Xi

24 Normalize Xk in Xi and Bj
25 Round and Normalize Xk in Xi and Bj

26 Unpack Xk to Xi and Bj
27 Pack Xi from Xk and Bj

43 Form jk Mask in Xi

The Shift Unit operates in a manner similar to the Boolean and Fixed Add Units. Control bits are entered in the unit at the time the instruction is issued. A timing sequence is initiated by the Scoreboard control, and results are held until released for transfer to the CPU registers.

Overall operation of this unit is described as follows according to the instruction groups.

20 and 21—Shift jk—These instructions shift the contents of CPU register Xi left or right a total of jk bit positions. The shift count comes directly from the jk field of the instruction.

Left shift is circular, such that the most significant bit is reinserted at the least significant bit position of the word for a shift of one place.

Right shift is end-off with sign extension.

22 and 23—Shift Bj—These instructions shift the contents of CPU register Xk left or right, placing the result in CPU register Xi. Xk and Xi may be the same register. The shift count is the absolute magnitude of the signed number in Bj made up of the sign bit and the lowest order six bits.

The direction of the shift is determined by the sign of Bj and the instruction. A positive sign in Bj causes the direction to be as "nominally" stated. In other words, instruction 22 states Shift Xk nominally left Bj places and will cause a left circular shift only if Bj is positive. Similarly, instruction 23 will cause a right end-off shift only if Bj is positive. If Bj is negative, the directions will reverse.

Note: These instructions are very convenient for scaling floating point numbers to align the binary point. For example, if the floating point number +067000—E11 were unpacked, the exponent would appear in a B register as −11 (octal). A 22 instruction specifying the X register and the shift count in the B register would shift the number correctly to the right 11 bit positions to align the binary point to the right of the least significant bit, or +000067.

Note: The floating point format is described in the next section of this Chapter on the ADD Unit.

24 and 25—NORMALIZE—These instructions determine the number of leading zeros in the coefficient of the floating point number located in the Xk register. The number may be positive or negative. (For negative numbers, the count of leading ones is determined.) The number of leading zeroes, excluding the sign bit, is the correct number of left shifts to normalize the number. This shift count is placed in CPU register Bj and is also used to control a following left shift.

26 and 27—PACK/UNPACK—These instructions are used to separate or couple the elements of a floating point number. The exponent is found in the CPU register Bj with the coefficient in a CPU X register and the combination also in a CPU X register.

These instructions provide a convenient means for converting between fixed point integers and floating point numbers.

43—MASK—This instruction forms a mask in CPU register Xi. The six-bit quantity jk entered directly from the instruction defines the number of ones in the mask as counted from the highest order bit in register Xi. This is a simple means of forming a contiguous string of ones in a 60-bit word for simple masking. This can be followed by a left-circular shift to place it correctly in the word.

The Shift Unit contains an advanced form of parallel shifting network. In older computers, shifting was accomplished using two registers and transferring between them with a single-bit displacement. This form of shifting required a variable number of shift cycles dependent on the number of places to be shifted. A number of awkward schemes for minimizing the number of shifts have been used. Particularly in floating point addition and in normalization or alignment processes, the shift operation is a significant consumer of time.

The 6600 Computer offered an opportunity, in its framework of separate functional units, to provide a shift apparatus which would not be variable in time. The principle of operation is based on six columns of logic, each column controlled by one of the bits of shift count. If the shift count bit is zero, the entire column is transferred "straight through" to the next column. If the shift count bit is one, the entire column is transferred to the next column, displaced by an amount corresponding to the specific weight or bit of shift count. This is shown graphically in Figure 50. One bit is shown shifted by three shift counts in octal: 00, 37, and 77.

The shift logic is made up as a very simple gating arrangement to accomplish the selection between no shift and a shift specified by SKn. A small section of the network is shown in Figure 51 giving, in this case, both the right

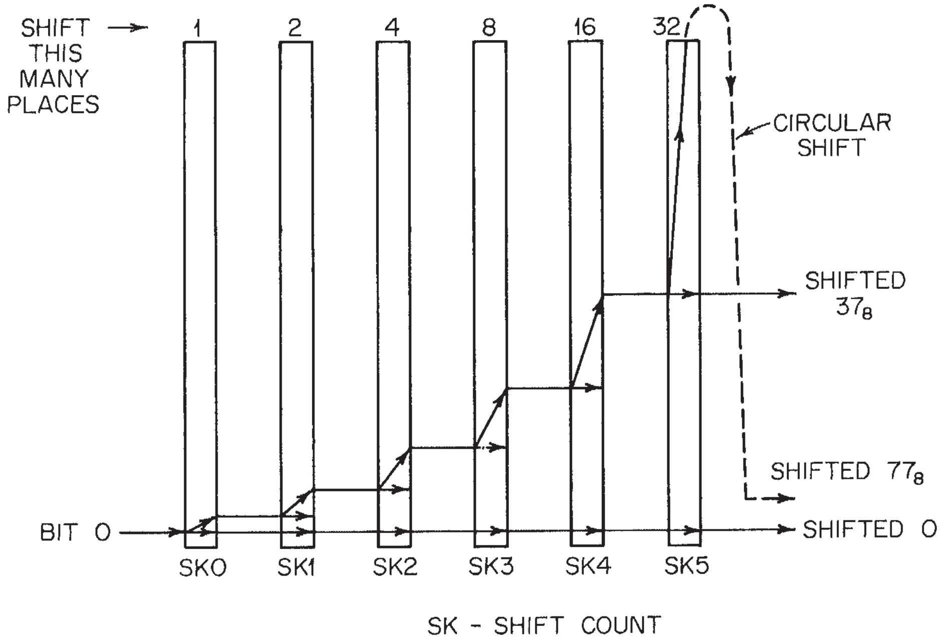

SHIFT THIS MANY PLACES
1
2
4
8
16
32
CIRCULAR SHIFT
SHIFTED 37_8
SHIFTED 77_8
BIT 0
SHIFTED 0
SK0
SK1
SK2
SK3
SK4
SK5
SK - SHIFT COUNT

FIGURE 50

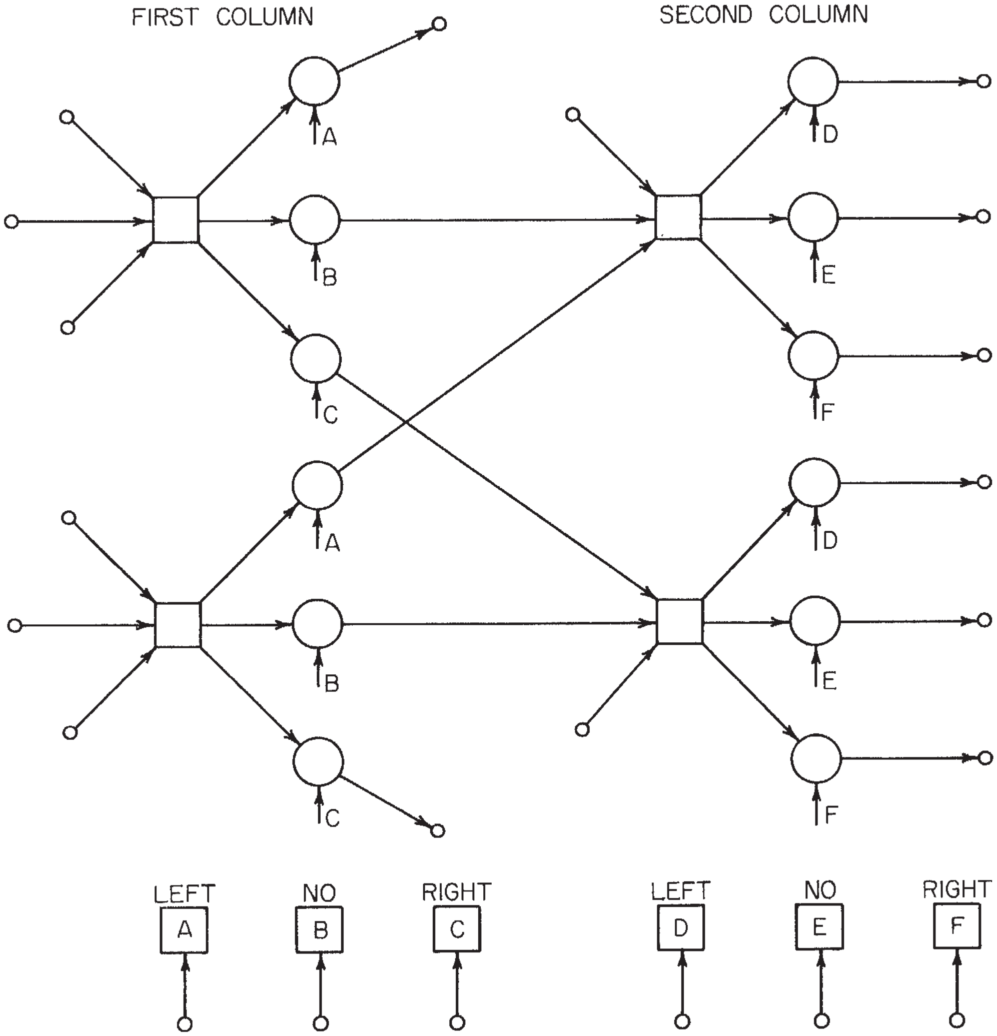

FIRST COLUMN
SECOND COLUMN
A
B
C
A
B
C
D
E
F
D
E
F
LEFT
A
NO
B
RIGHT
C
LEFT
D
NO
E
RIGHT
F

FIGURE 51

and left shifts. This assumes that only two bits of network are contained on a module.

If the network is contained in the manner shown, two columns are held in a single set of modules. Bits are placed in the modules so that physically adjacent bits are separated by an amount equal to the shift accomplished between the first and second columns within the module. Shifts external to the module are, of course, easily wired to the correct input pin on the correct module.

Note that the network requires a fan-in to the OR element of three and a fan-out of three. Each column uses exactly the same number of logic elements. A single module type is used, except for some special end cases. The right shift is complicated by the need to extend the sign bit. However, this extension is accomplished by generating the sign bit ANDed with the column controls in the quantity needed for each column, as follows.

Right shift count $SK_0 = 1$ AND Sign Bit— 1 needed.
$SK_1 = 1$ AND Sign Bit— 2 needed.
$SK_2 = 1$ AND Sign Bit— 4 needed.
$SK_3 = 1$ AND Sign Bit— 8 needed.
$SK_4 = 1$ AND Sign Bit—16 needed.
$SK_5 = 1$ AND Sign Bit—32 needed.

These signals are entered into the Shift network by fan-out circuits as needed. Note that the first columns need the least such terms. Therefore, fan-out requirements for the last columns are easily achieved without interfering with the timing.

The Shift network is assembled on QF modules with three bits of two columns on each module. Four special KF modules are substituted for four QF for special end cases. The total number of modules, excluding controls, in the Shift network is 60, a small number indeed, considering that only one minor cycle is needed to pass through the network, accomplishing any shift ranging from one to 60, either right or left.

NORMALIZE NETWORK

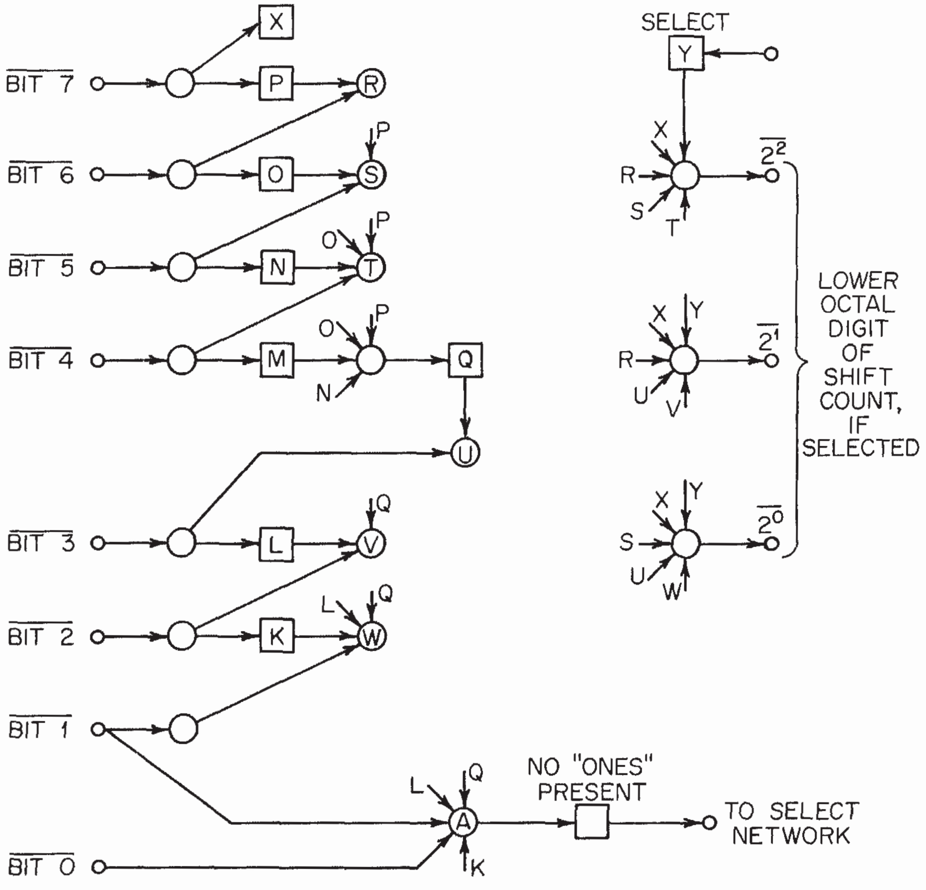
For instructions 24 and 25, an additional network called the *normalize network* is used. This network determines the number of bits of left shift necessary to normalize and stores this six-bit number in the shift control counter SK. A left shift is then performed in the shift network under control of SK.

To simplify the normalize network, the number to be normalized is first forced positive on entry to the Shift Unit Input Register. If complementing is required, a "complement" flag is set.

The coefficient portion of the floating point number to be normalized is examined in six groups of eight bits each. Two immediate determinations are made.

1. The highest order "one" in each group of eight bits.
2. The highest order group with a "one."

The first determination is made simultaneously in six identical modules, shown in Figure 52. The second determination is made similarly in one extra module. The result of the second test is an octal digit defining the number of eight-bit shifts which must be taken. The result also selects which of the other six outputs should be taken as the number of single-bit shifts to be taken. These two quantities are merged to form the six-bit shift count in SK.



$\overline{\text{BIT 7}}$
$\overline{\text{BIT 6}}$
$\overline{\text{BIT 5}}$
$\overline{\text{BIT 4}}$
$\overline{\text{BIT 3}}$
$\overline{\text{BIT 2}}$
$\overline{\text{BIT 1}}$
$\overline{\text{BIT 0}}$
X
P
R
O
S
N
T
M
Q
U
L
V
K
W
A
SELECT
Y
$\overline{2^2}$
$\overline{2^1}$
$\overline{2^0}$
LOWER OCTAL DIGIT OF SHIFT COUNT, IF SELECTED
NO "ONES" PRESENT
TO SELECT NETWORK

FIGURE 52

The first test is accomplished by a simple test of each bit of eight compared with the bits more significant in the group. A three-bit number is formed directly from the logic, and a zero test of all bits is also formed.

During a normalize shift, the floating point coefficient is shifted left accompanied by a corresponding reduction of the exponent value. It is possible to cause underflow during the normalize operation.

Rounding is accomplished by appending a "one" to the coefficient at the beginning of the normalize shift. This assumes that the number being rounded is single precision, which is, of course, precisely the way it appears to the Shift Unit. The round bit is forced into the shift network and appears as one bit "to the right" of bit position 0 during the left circular shift.

The remainder of the logic in the Shift Unit is devoted to the Pack, Unpack, and Mask instructions and to the determination of the direction of the shift, whether left or right.

The mask operation utilizes a forced "one" in bit 59, the highest order bit, similar to the round operation but followed by a right shift rather than left.

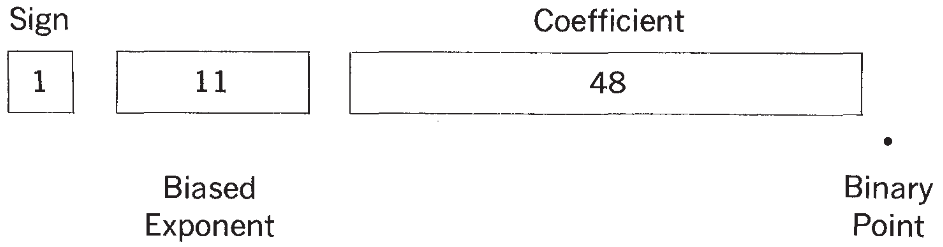
Note also that this functional unit does not utilize a temporary result register since the input registers are held until release. As a result, the networks are allowed to stabilize before the result is sampled and the unit released.

E. ADD UNIT

The Add Unit is designed to perform floating point addition and subtraction. These computations may be made in rounded single precision or unrounded single and double precision. Total execution time is 400 nanoseconds including one minor cycle for placing the result in the selected CPU register. The following instructions are executed in the Add Unit.

30 Floating Sum of Xj and Xk to Xi
31 Floating Difference of Xj and Xk to Xi
32 Floating Double Precision Sum of Xj and Xk to Xi
33 Floating Double Precision Difference of Xj and Xk to Xi
34 Rounded Floating Sum of Xj and Xk to Xi
35 Rounded Floating Difference of Xj and Xk to Xi

Floating point numbers utilize the following format.



One's complement number representation is used on all numbers within the 6600, including the exponent and coefficient of the floating point numbers.

A simple description of this number representation is given in an earlier section of this Chapter on the Fixed Add Unit.

The use of the integer representation of the coefficient is a particularly interesting convenience. This allows short fixed-point integers held in the Index registers to be simply converted into floating-point numbers. A pack instruction can be used to introduce the exponent bias without any shift or exponent adjustment needed. The exponent is a signed binary integer operating on a base two, as follows.

Floating-point number, $K \cdot 2^e$
where K defines the 48-bit one's complement coefficient with its sign being the sign bit of the entire number,
e defines the 11-bit signed exponent.

The bias is applied to the exponent in order to place the "zero" exponent in the middle of the range of numbers. The following numbers are examples, given in octal.

0000 000 . . . 000. = $+0 \cdot 2^{-1777}$, a small number indeed.
2000 000 . . . 001. = $+1 \cdot 2^{+0}$, the integer 1.
5777 777 . . . 712. = $-65 \cdot 2^{+0}$.

The last example shows that the exponent is complemented if the number is negative. The net effect of the bias and the negative-number treatment on the exponent is to maintain a consistent ascending order to all numbers from smallest to largest. This holds true for fixed-point and floating-point numbers.

The exponent bias is very simply applied to the "assembled" floating-point number by reversing the exponent sign bit. Therefore, manipulation of exponents within the functional units must also make this reversal. For clarity, the following cases are given.

2000 xx . . . x = $+x \cdot 2^0$
5777 xx . . . x = $-\bar{x} \cdot 2^0$
2016 xx . . . x = $+x \cdot 2^{16}$
1735 xx . . . x = $+x \cdot 2^{-42}$

Provision is made for the treatment of overflow conditions in this format. Three cases are important. These are the infinite case, the indefinite case, and the underflow case.

Any result with an exponent so large that it reaches or exceeds the upper limit of 3777 (positive) or 4000 (negative) is treated as an infinite quantity. Recognition of this exponent in input operands can produce an error exit, if selected.

The use of infinity, zero, or indefinite operands may produce an indefinite result, as shown in the following table. An exponent of octal 1777 and a zero coefficient are packed in this case. An error exit may occur on recognition of this quantity or its complement as an input operand, if selected.

Any result with an exponent less than or equal to the lower limit of octal 0000 (positive) or 7777 (negative) is treated as zero. A result which reaches a value of zero exponent, but with a nonzero coefficient is left that way. Any following usage, however, considers the number zero for purposes of infinity and indefinite.

The Add Unit is an example of one key design advantage arising from separate functional units. This allows a design uncluttered by any functions other than the floating addition and subtraction. The resultant design provides a minimum execution time for this function.

Addition of two floating point numbers requires that the binary points be aligned. The approach taken in this unit is to hold the number with

TABLE V Nonstandard Floating Point Arithmetic

The following is a tabulation of operations (Add, Subtract, Multiply, Divide) using various combinations of operands.

KEY:

Operands	Results
+0 = 0000 X . . . X	0 = 0000 0 . . . 0
−0 = 7777 X . . . X	IND = 1777 0 . . . 0
+∞ = 3777 X . . . X	+∞ = 3777 0 . . . 0
−∞ = 4000 X . . . X	−∞ = 4000 0 . . . 0
+IND = 1777 X . . . X	
−IND = 6000 X . . . X	
W = Any word except ±∞, ±IND	
N = Any word except ±∞, ±IND, or ±0	

ADD

Xi = Xj + Xk

(Instructions 30, 32, 34)

		Xk			
		W	+∞	−∞	±IND
Xj	W	W	+∞	−∞	IND
	+∞	+∞	+∞	IND	IND
	−∞	−∞	IND	−∞	IND
	±IND	IND	IND	IND	IND

SUBTRACT

Xi – Xj – Xk

(Instructions 31, 33, 35)

		Xk			
		W	+∞	−∞	±IND
Xj	W	W	−∞	+∞	IND
	+∞	+∞	IND	+∞	IND
	−∞	−∞	−∞	IND	IND
	±IND	IND	IND	IND	IND

TABLE V (continued)

MULTIPLY

$Xi = Xj * Xk$

(Instructions 40, 41, 42)

		Xk						
		+N	−N	+0	−0	$+\infty$	$-\infty$	±IND
Xj	+N	+N	−N	0	0	$+\infty$	$-\infty$	IND
	−N	−N	+N	0	0	$-\infty$	$+\infty$	IND
	+0	0	0	0	0	IND	IND	IND
	−0	0	0	0	0	IND	IND	IND
	$+\infty$	$+\infty$	$-\infty$	IND	IND	$+\infty$	$-\infty$	IND
	$-\infty$	$-\infty$	$+\infty$	IND	IND	$-\infty$	$+\infty$	IND
	±IND	IND	IND	IND	IND	IND	IND	IND

DIVIDE

$Xi = Xj/Xk$

(Instructions 44, 45)

		Xk						
		+N	−N	+0	−0	$+\infty$	$-\infty$	±IND
Xj	+N	+N	−N	$+\infty$	$-\infty$	0	0	IND
	−N	−N	+N	$-\infty$	$+\infty$	0	0	IND
	+0	0	0	IND	IND	0	0	IND
	−0	0	0	IND	IND	0	0	IND
	$+\infty$	$+\infty$	$-\infty$	$+\infty$	$-\infty$	IND	IND	IND
	$-\infty$	$-\infty$	$+\infty$	$-\infty$	$+\infty$	IND	IND	IND
	±IND	IND	IND	IND	IND	IND	IND	IND

larger exponent, while shifting the number with smaller exponent to the right. The amount of shift is determined by subtracting the smaller exponent from the larger.

As an example, add the following octal numbers:

Xj = 0 2005 0 . . . 05244. $(+5244 \cdot 2^{5})$
Xk = 0 2016 0 . . . 07305. $(+7305 \cdot 2^{16})$

The number in Xk is held, while the number in Xj is shifted to the right a total of 11 (octal) places.

The new positions of these coefficients can be shown as if in registers of double length, as follows:

Xk	0 . . . 07305.0 0
Xj (Shifted)	0 . . . . . . 05.2440 . . . 0
Sum	0 . . . 07312.2440 . . . 0

The binary point as shown in the sum is in the same position as the number in Xk. Therefore, the single precision result of this addition would be the upper half as follows:

Single precision sum 0 2016 0 . . . 07312. ($7312 \cdot 2^{16}$)

For the case of double precision solution, the lower half can also be taken, but the exponent must be reduced to account for moving the binary point to the right 48 places. The exponent for the lower half is thus determined by reducing the upper half exponent by 60 (octal). This is accomplished by first removing the bias, as follows:

$$\text{Octal exponent} = 0016 - 60 = -42$$

This is described in the 11-bit exponent field, without bias, as 3735. Applying the bias by simply reversing the exponent sign bit produces the exponent of 1735. Therefore, the double precision result of the above addition would be the lower half, as follows:

Double precision sum 0 1735 2440 . . . 0. ($2440 \ldots 0 \cdot 2^{-42}$)

This is, of course, something of a misnomer since the double precision sum is, in fact, both the upper and lower halves. However, the two separate halves are now placed in a form in which subsequent double precision operations are entirely valid.

Other circumstances possible during an Addition or Subtraction include *overflow* and *underflow*. An example of an overflow case is given below.

Xj 0 2032 7700 . . . 0. ($+7700 \ldots 0 \cdot 2^{32}$)
Xk 0 2032 7760 . . . 0. ($+7760 \ldots 0 \cdot 2^{32}$)

No alignment is necessary since both exponents are equal. The addition obviously causes an overflow, as follows.

$$\begin{array}{lr} \text{Xj coef} & 7700 \ldots 0. \\ \text{Xk coef} & 7760 \ldots 0. \\ \hline & 17660 \ldots 0. \end{array}$$

This is the maximum overflow possible on coefficients in the Add Unit and is, therefore, detected and corrected before returning the result to the CPU register Xi. The correction is very simply a right shift of one-bit position and an increase by one of the result exponent. This will produce the following result.

Sum 0 2033 7730 . . . 0. ($+7730 \ldots 0 \cdot 2^{33}$).

Because the exponent is increased during this correction, it is also possible to "generate" an infinity condition.

For the *underflow* circumstance, the following example is given. The case arises only in the use of the Add Unit for double precision solution. Therefore, the example is a "lower half" result.

Xj 0 0040 00 . . . . 72. $(+\ 72 \cdot 2^{-1737})$
Xk 0 0032 00 . . . 332. $(+332 \cdot 2^{-1745})$

The result of the coefficient alignment and sum is accomplished thus.

Xj coef	0 00 . . . 072.0 0
Xk coef	0 00 . . . . 03.320 . . 0
Sum	0 00 . . . 075.320 . . 0

In taking the lower half, however, the reduction of the exponent "underflows" on the 11-bit exponent field.

Octal exponent $-1737 - 60 = -2017$

The proper exponent, in the case of an underflow, is all zeroes.

EXPONENT CALCULATION

The Add Unit begins its operation by testing the relationship of the exponents. The test is conducted to determine which is larger, and then to produce a shift count for right shifting the number with the smaller exponent. The total exponent calculations required are:

1. select the coefficient with the smaller exponent for entry to the right shift network;
2. form an absolute magnitude shift count representing the difference between exponents;
3. select the larger exponent as the exponent for use with upper-half result;
4. subtract 60 (octal) from upper-half exponent for use as lower-half exponent;
5. adjust upper- and lower-half exponents in case of overflow;
6. form all zeroes exponent for lower half in case of underflow.

The first output needed from the exponent subtraction is the choice of the coefficient to be shifted. Shortly thereafter, the bits of shift count are required to control the shift network. For these requirements, the exponents are extracted from the full number and complemented if the number was negative. The bias is removed by complementing the exponent sign bit. The two numbers are then placed in a 12-bit subtract network (the exponent sign bit is copied into the 12th bit). Five cases are of interest.

1. Both exponents positive and equal.
2. Both positive and unequal.
3. Signs unlike.
4. Both negative and unequal.
5. Both negative and equal.

The choice of smaller exponent for signs unlike is, of course, simply the negative exponent. The case of exponents equal is also no contest; an arbitrary choice is satisfactory. The only significant calculation is therefore on the unequal exponents of like sign.

The larger of two numbers can be determined in a one's complement subtract rather easily by examining the end-around-borrow. A network is built similar to the subtractive network used in the Fixed Add Unit, except only 12 bits long. The end-around-borrow for this case can be derived assuming one exponent is X and the second exponent is the complement of Y. In other words, the add network forms the sum of X and minus Y.

From the discussion in the section on the Fixed Add Unit, it was seen that borrows are *generated* by $\overline{X}n\overline{Y}n$ terms, and *passed* by $\overline{X}n + \overline{Y}n$ terms. This means that a borrow must be generated but may then be passed to the left. Looking at this property with regard to the test for the larger exponent, it can be seen that the end-around-borrow is sensitive to the relative size.

The following examples describe the case.

```
         ┌────←B     |←────┐
   X =   │000 001 111 000  │      +0170 (Octal)
  -Y =   │111 100 111 111  │      -0300
         └─────────────────┘      ─────
 Sum =    111 110 110 111         -0110
```

In this first example, a borrow is generated in the fifth position from the left and passed all the way, end-around, finally stopping at the fourth position from the right.

Reversing the above numbers,

```
              | ← BBB
   X = 000 011 000 000      +0300 (Octal)
  -Y = 111 110 000 111      -0170
       ───────────────      ─────
 Sum = 000 001 001 000      +0110
```

Duplicating these numbers, except in negative form, the following examples are given.

```
              | ← BBB
   X = 111 110 000 111      -0170 (Octal)
  -Y = 000 011 000 000      +0300
       ───────────────      ─────
 Sum = 000 001 001 000      +0110
```

```
         ┌────←B     |←────┐
   X =   │111 100 111 111  │      -0300 (Octal)
  -Y =   │000 001 111 000  │      +0170
         └─────────────────┘      ─────
 Sum =    111 110 110 111         -0110
```

From this exercise it can be seen that exponent Y is always larger, that is more positive, than exponent X if an end-around-borrow is generated. The absence of an end-around-borrow indicates either that the two exponents are equal or that exponent X is larger than exponent Y.

The above can be described in more rigorous terms. An interesting quirk of this logic is that for exponents with unlike signs, the existence of an end-around-borrow specifies not Y but X as the larger. As a result, the choice of coefficient to be shifted is found by the exclusive OR of End-Around-Borrow and Exponent Signs alike.

Two instructions provide for rounding the single precision result of the floating add unit. If both input operands are normalized, they may be considered to be larger in absolute magnitude by one-half of the least significant bit. Since the add network is essentially double length, this may easily be forced into the network by appending an extra bit to each operand.

However, it is convenient to deal with unnormalized numbers and especially mixtures of normalized numbers and constants. A constant integer, for example, should be treated consistently as an integer. Since Add and Subtract are the only functions sensitive to this, the round mechanism for the Add Unit is especially built, as follows.

1. A round bit is attached at the right end of both operands if:
   a. both operands are normalized, or
   b. the operands have unlike signs.
2. A round bit is attached at the right end of the operand with the larger exponent for all other cases.

RIGHT SHIFT NETWORK

Having picked the coefficient with smaller exponent and determined the proper shift count, the next major step in the Add Unit is the alignment shift. Seven bits of shift count are determined by the exponent calculation. If the exponent difference is greater than the seven-bit count, the shifted coefficient essentially "disappears" to the right of the add network.

The right shift network is a parallel shift network similar to the Shift Unit. Since it is used strictly for right shifts, however, the design can be tightened with some speed improvement. A small section of this network is shown in Figure 53.

In this network, each node is logically active in performing the right shift. The following derivation shows the method.

$$F = T + JKS + LM\bar{S}$$
$$G = \bar{T} + JKS + LM\bar{S}$$

Assuming term S is the control term RIGHT SHIFT 2 places, the combination term JK can be said to be *shifted* right to terms F and G. If no Right Shift 2 is desired, the combination term LM can be said to be *passed through* to terms F and G.

Assuming term T is the control term RIGHT SHIFT 4 places, the term F will be the pass through term or will be a one during the shift. Similarly, the term G will be the Right Shift 4 term or will be a one during pass through. The effect is to combine two terms at the next pair of nodes, such as P and Q. One input to P and Q is the pass through term, in this case F. The other input

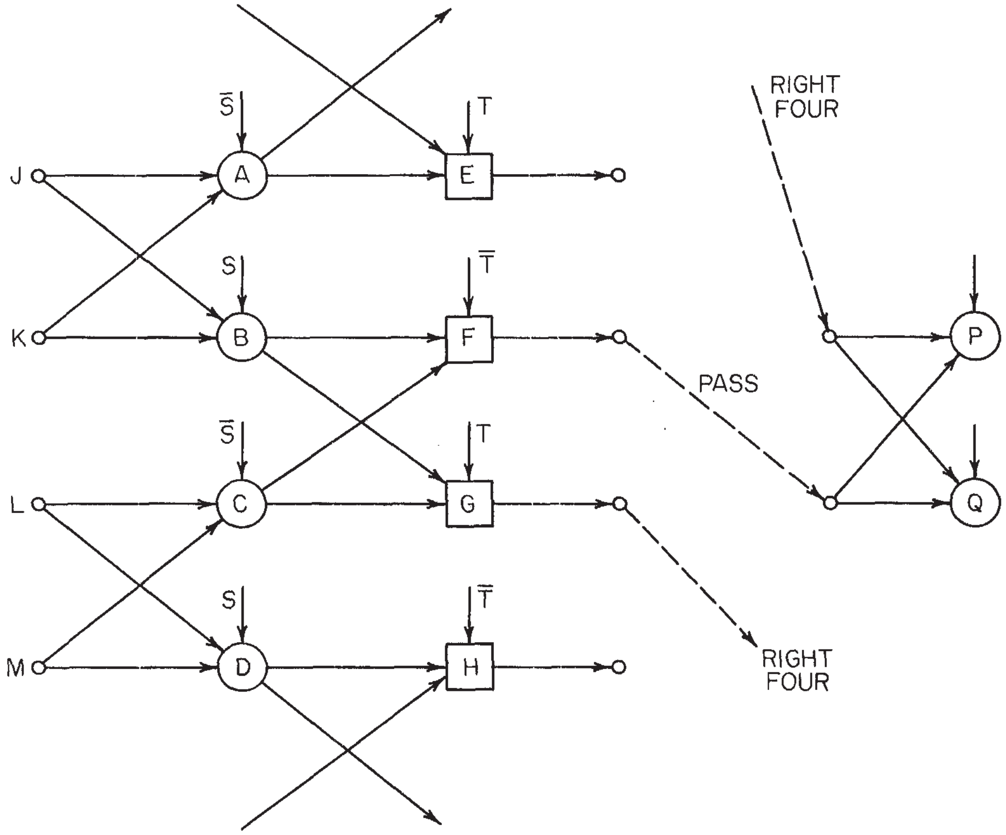


FIGURE 53

to P and Q is the right shift term, similar to term G but displaced "left" of G by four bit positions.

Note that the entering terms JK and LM can be formed in exactly the same manner as the entering terms for P and Q. In this case, JK must be two bit positions "left" of LM in order that S be the control for right shift two places.

This network is a minimum hardware version of a fully parallel right shift network and is also a minimum time version. Only seven inverters are needed for the seven-bit shift count used in the Add Unit. This is shown in block diagram form in Figure 54 (page 86).

The output of the right shift network is 96 bits plus an additional two sign bits. The result of the shift involves a right shift of the selected sign bit on a "background" of sign bits. Sign bits are, therefore, duplicated into each rank of shift network as needed.

The adder network is a 98-bit parallel version similar to that discussed in the Fixed Add Unit. A slightly different packaging combination is used, however, showing the intimacy of the logic and the package once again. In this case two bits of the adder "entry" and "output" are contained on one module, as opposed to three bits on the Fixed Add Unit. (See Figure 55.)

The adder "entry" is made up of the coefficient whose exponent was larger and the output of the shift network. Since the last rank of the shift network causes a right shift of 64 places, the network output is made up of a

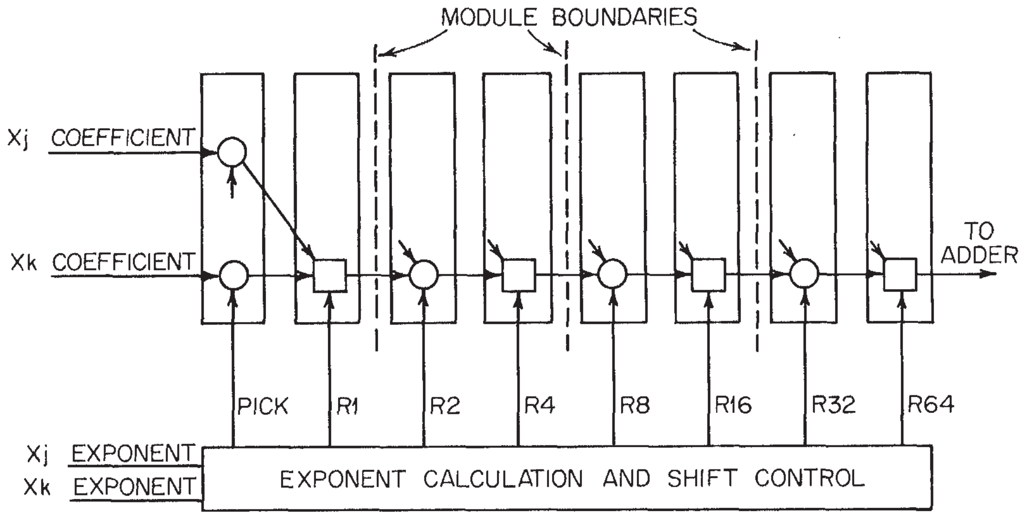

MODULE BOUNDARIES
Xj COEFFICIENT
Xk COEFFICIENT
TO ADDER
PICK
R1
R2
R4
R8
R16
R32
R64
Xj EXPONENT
Xk EXPONENT
EXPONENT CALCULATION AND SHIFT CONTROL

FIGURE 54

sign bit or a bit from the second coefficient, right shifted. These quantities are combined to form the necessary initial borrow generation and borrow pass terms. The borrows are then successively combined in sets of three, three, and six as shown in the following equations.

$$
\begin{aligned}
B_1 &= \bar{X}_0\bar{Y}_0 + (\bar{X}_0 + \bar{Y}_0)B_0 \\
B_2 &= \bar{X}_1\bar{Y}_1 + \bar{X}_0\bar{Y}_0(\bar{X}_1 + \bar{Y}_1) + (\bar{X}_0 + \bar{Y}_0)(\bar{X}_1 + \bar{Y}_1)B_0 \\
&= D_1 + E_1B_0,
\end{aligned}
$$

where D_1 – Borrow generation from bit 1,
E_1 – Borrow pass for bits 0 and 1.

At the next borrow "layer," all D and E terms can be grouped in three's.

$$
\begin{aligned}
B_2 &= D_1 + E_1B_0 \\
B_4 &= D_3 + D_1E_3 + E_1E_3B_0 \\
B_6 &= D_5 + D_3E_5 + D_1E_3E_5 + E_1E_3E_5B_0 \\
&= F_5 + G_5B_0.
\end{aligned}
$$

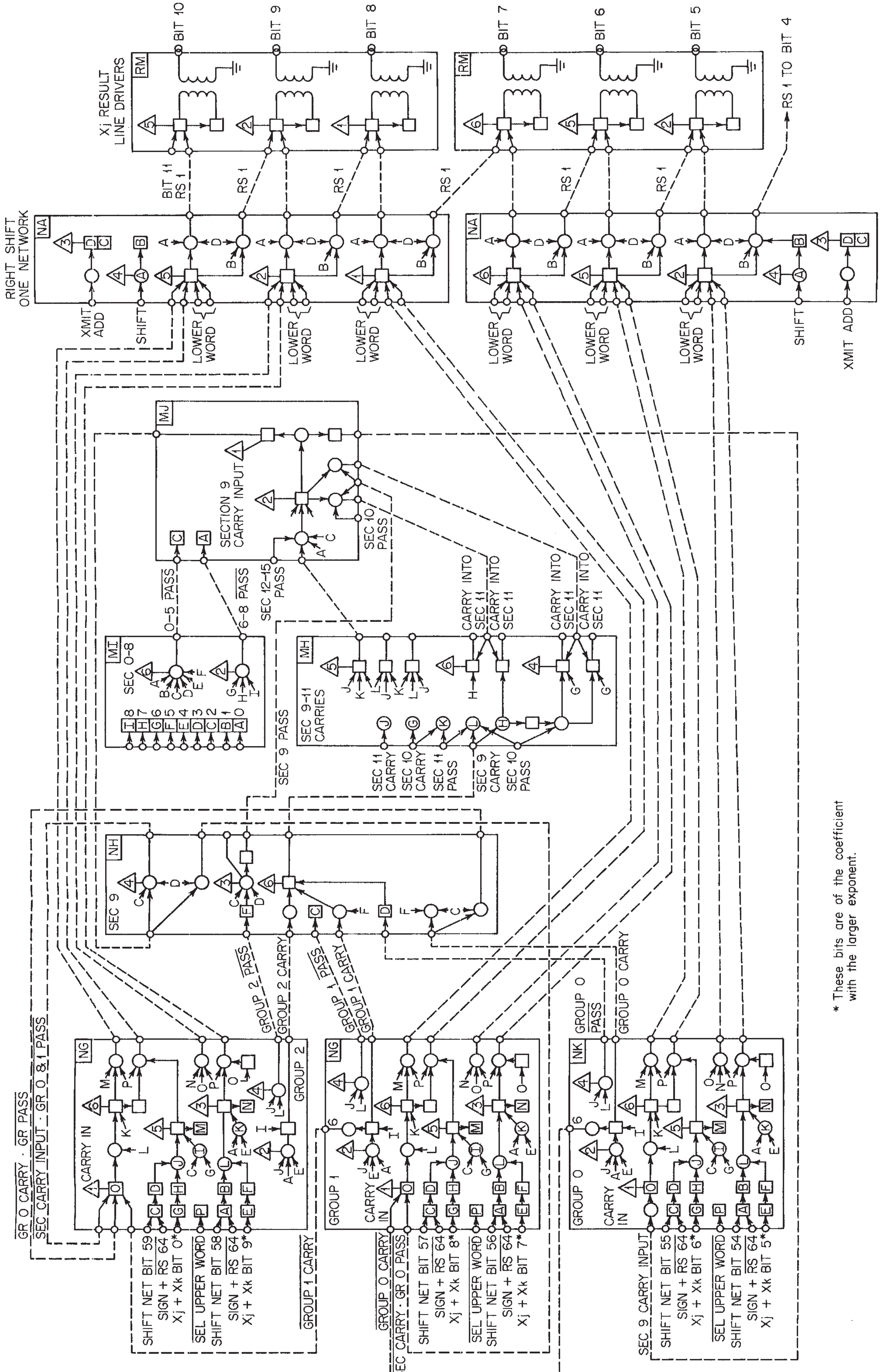
Similarly, the next borrow "layer" combines in groups of three.

$$
\begin{aligned}
B_6 &= F_5 + G_5B_0 \\
B_{12} &= F_{11} + F_5G_{11} + G_5G_{11}B_0 \\
B_{18} &= F_{17} + F_{11}G_{17} + F_5G_{11}G_{17} + G_5G_{11}G_{17}B_0 \\
&= J_{17} + K_{17}B_0.
\end{aligned}
$$

Finally, these terms may be grouped in six terms, as follows:

$$
\begin{aligned}
B_0 = B_{108} = J_{107} &+ J_{89}K_{107} + J_{71}K_{89}K_{107} + J_{53}K_{71}K_{89}K_{107} + J_{35}K_{53}K_{71}K_{89}K_{107} \\
&+ J_{17}K_{35}K_{53}K_{71}K_{89}K_{107}.
\end{aligned}
$$

$$
\begin{aligned}
B_{90} = J_{89} &+ J_{71}K_{89} + J_{53}K_{71}K_{89} + J_{35}K_{53}K_{71}K_{89} + J_{17}K_{35}K_{53}K_{71}K_{89} \\
&+ J_{107}K_{17}K_{35}K_{71}K_{89}.
\end{aligned}
$$


RIGHT SHIFT ONE NETWORK
NA
XMIT ADD
SHIFT
LOWER WORD
BIT 11 RS 1
RS 1
Xj RESULT LINE DRIVERS
RM
BIT 10
BIT 9
BIT 8
BIT 7
BIT 6
BIT 5
RS 1 TO BIT 4
MJ
SECTION 9 CARRY INPUT
SEC 10 PASS
SEC 12-15 PASS
0-5 PASS
6-8 PASS
MI
SEC 0-8
MH
SEC 9-11 CARRIES
SEC 11 CARRY
SEC 10 CARRY
SEC 11 PASS
SEC 9 CARRY
SEC 10 PASS
CARRY INTO SEC 11
SEC 9 PASS
NH
SEC 9
GROUP 2 PASS
GROUP 2 CARRY
GROUP 1 PASS
GROUP 1 CARRY
GROUP 0 PASS
GROUP 0 CARRY
NG
NK
GR 0 CARRY · GR PASS
SEC CARRY INPUT · GR 0 & 1 PASS
CARRY IN
GROUP 2
GROUP 1
GROUP 0
SHIFT NET BIT 59
SIGN + RS 64
Xj + Xk BIT 0*
SEL UPPER WORD
SHIFT NET BIT 58
Xj + Xk BIT 9*
GROUP 1 CARRY
GROUP 0 CARRY
SEC CARRY · GR 0 PASS
SHIFT NET BIT 57
Xj + Xk BIT 8*
SHIFT NET BIT 56
Xj + Xk BIT 7*
SEC 9 CARRY INPUT
SHIFT NET BIT 55
Xj + Xk BIT 6*
SHIFT NET BIT 54
Xj + Xk BIT 5*

* These bits are of the coefficient with the larger exponent.

FIGURE 55

$$B_{72} = J_{71} + J_{53}K_{71} + J_{35}K_{53}K_{71} + J_{17}K_{35}K_{53}K_{71} + J_{107}K_{17}K_{35}K_{53}K_{71} + J_{89}K_{107}K_{17}K_{35}K_{53}K_{71}$$

$$B_{54} = J_{53} + J_{35}K_{53} + J_{17}K_{35}K_{53} + J_{107}K_{17}K_{35}K_{53} + J_{89}K_{107}K_{17}K_{35}K_{53} + J_{71}K_{89}K_{107}K_{17}K_{35}K_{53}.$$

$$B_{36} = J_{35} + J_{17}K_{35} + J_{107}K_{17}K_{35} + J_{89}K_{107}K_{17}K_{35} + J_{71}K_{89}K_{107}K_{17}K_{35} + J_{53}K_{71}K_{89}K_{107}K_{17}K_{35}.$$

$$B_{18} = J_{17} + J_{107}K_{17} + J_{89}K_{107}K_{17} + J_{71}K_{89}K_{107}K_{17} + J_{53}K_{71}K_{89}K_{107}K_{17} + J_{35}K_{53}K_{71}K_{89}K_{107}K_{17}.$$

Note that only the *Borrow Generation* terms are necessary and that all six terms involve a maximum of six elements of AND or OR. The network shown is sufficient for an add of 108 bits. However, the ADD Unit requires only 96 bits for the double length add, one overflow bit and one sign bit. The operands are positioned in this adder such that the overflow bit is obtained "early." This determination must be made in order to make a corrective shift of one place and an increase by one of the exponent for the overflow case.

Output of the adder network is sampled directly into the transmitter circuits, controlled by the Scoreboard. Choice of the upper half or lower half is made just in advance of the transmitters. The right shift of one place required after an overflow is also applied to the lower-half coefficient and the adjustment to the lower-half exponent.

F. MULTIPLY UNIT

Two identical Multiply Units are included in the 6600 Central Processor. While this is small extravagance, the use of multiplication, particularly in multiple precision computation, represents a large percentage of time. Each of these units takes 1000 nanoseconds as compared to the 400 nanoseconds for the Add Unit. Each Multiply Unit can execute the following instructions.

40—Floating Product of Xj and Xk to Xi.
41—Rounded Floating Product of Xj and Xk to Xi.
42—Floating Double Precision Product of Xj and Xk to Xi.

As in the Add Unit, both single and double precision results are produced. These are the upper and lower halves of the product resulting from the multiplication of two single precision operands. The two halves must be obtained separately, as in the Add. With two Multiply Units, however, the two halves can be obtained on two successive minor cycles, a cost of 1000 nanoseconds for the first half and an effective cost of 100 nanoseconds for the second half. This overlap is easily obtained using two X registers for the results which are different from the X registers holding the input operands.

The double precision result of floating multiplication shown below is a double length coefficient with an exponent equal to the sum of the original exponents.

$$\begin{array}{rr}
\text{Xj} = 0\ \ 2001\ \ 0\ldots\ldots 7. & (+\quad 7\cdot 2^{1})_8 \\
\text{Xk} = 0\ \ 2002\ \ 0\ldots 102. & \text{x}(+102\cdot 2^{2})_8 \\
\hline
0\ \ 2003\ \ 0\ldots\ldots\ldots 0\ \ 0\ldots 716. & (+716\cdot 2^{3})_8
\end{array}$$

The exponent for this product is correct for the binary point at the far right as shown. The single precision result, however, takes the upper-half coefficient. The exponent must be increased by 60 octal in that case. The results of the above multiplication produce the following single and double precision results.

$$\begin{array}{lll}
\text{Single} & \text{Xi} = 0\ \ 2063\ \ 0\ldots\ldots 0. & (+\quad 0\cdot 2^{63})_8. \\
\text{Double} & \text{Xi} = 0\ \ 2003\ \ 0\ldots 716. & (+716\cdot 2^{3})_8.
\end{array}$$

It can be seen that the use of unnormalized arithmetic tends to force use of multiple precision. Double precision hardware is eminently desirable for unnormalized arithmetic. Normalized arithmetic, on the other hand, remains satisfactory for a single length coefficient. In multiplication of two normalized numbers, the result may "lose" normalization, as in the following example.

$$\begin{array}{lll}
 & \text{Xj} = 0\ \ 2101\ \ 460\ldots 0. & (+460\ldots 0\cdot 2^{101})_8 \\
 & \text{Xk} = 0\ \ 2020\ \ 400\ldots 0. & (+400\ldots 0\cdot 2^{20})_8 \\
\cline{2-3}
\text{Single} & \text{Xi} = 0\ \ 2201\ \ 230\ldots 0. & (+230\ldots 0\cdot 2^{201})_8
\end{array}$$

If the original operands were normalized, this movement of one bit position from "normal" is the maximum amount possible in the multiply operation. Therefore, the Multiply Units are designed to make the correction to the following normalized result.

$$\text{Single}\qquad \text{Xi} = 0\ \ 2200\ \ 460\ldots 0.\ (+460\ldots 0\cdot 2^{200})_8.$$

This exponent is decreased by one along with the single place left shift. The unit examines the input operands to determine if they are normalized. The above single place normalization of the result is activated *only* if both input operands were normal.

MULTIPLY METHODS

There are a number of schemes for multiplying two binary numbers, ranging in complexity. The simplest form using a parallel adder requires that the multiplier and partial product be shifted to the right relative to the multiplicand. The effect is easily seen in the example on page 90.

Multiplicand	1234
Multiplier	2143
Partial Product	3724
Partial Product	5160
Partial Product	1234
Partial Product	2470
Final Product	2671124

This example is given in octal form but should serve to show the method. Actually, the binary multiplier performs an addition and a shift for each bit of multiplier, rather than for each octal digit. In the above example, this would require twelve additions and shifts.

One scheme for improving the speed of multiplication is to separate the multiplier in half, performing the partial products simultaneously with a final addition. The example above is shown below using two halves.

Multiplicand	1234	1234
Multiplier	2100	0043
Partial Product	123400	3724
Partial Product	2470000	005160
	2613400	0055524
	0055524	
Final Product	2671124	

The time taken to accomplish the final product by this second method can be compared to the simple method as described in the following equations.

$T_1 = n(A + S)$,
where T_1 = multiplication time, simple case,
n = number of bits in multiplier,
A = time for a single addition,
S = time for a single place shift.

$T_2 = n/2(A + S) + A$,
where T_2 = multiplication time, two halves.

Carrying this type of scheme to more levels, the "add and shift" times can be reduced, but with an increase in the final additions to complete the final merged product. This can be defined in the following way.

$T_3 = \frac{n}{m}(A + S) + (m - 1)A$,
where T_3 = multiplication time, "many" adders,
m = number of separate adders.

Even this can be reduced some by combining the final merge in a "tree" of additions, as in the following example.

```
Multiplicand       1234          1234          1234          1234
Multiplier         2000          0100          0040          0003
                ---------     ---------     ---------     ---------
Partial Product  2470000   ┌─── 123400         51600   ┌───── 3724
                  123400 ←─┘                    3724 ←─┘
                 -------                       -----
                 2613400   ┌──────────────── 55524
                   55524 ←─┘
                 -------
Final Product    2671124
```

This shows that only two additions, not three as defined by the equation for T_3, are required for the final product.

Obviously, the examples are showing a diminishing return from this whole scheme. Figure 56 shows a plot of these schemes for several assumed add and shift times and takes into account "tree" techniques for the final additions. This is plotted for a multiplier of 48 bits, as needed for the 6600.

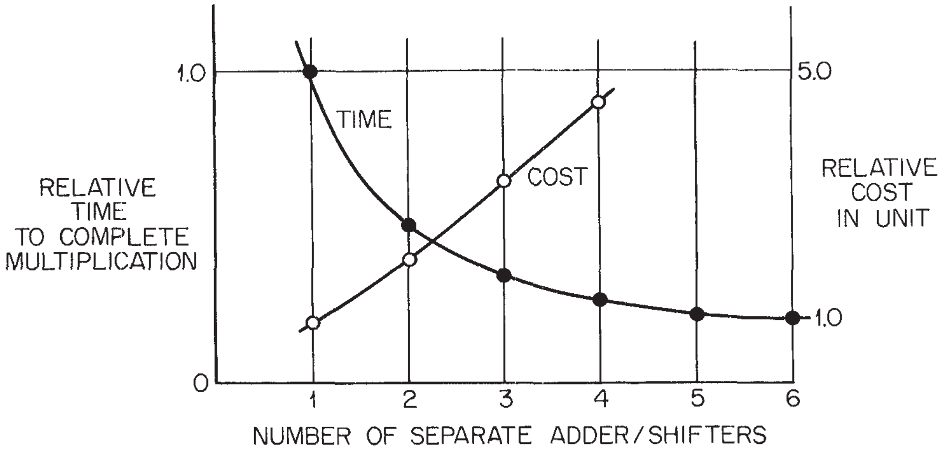


FIGURE 56

Another scheme for speeding multiplication arises from a technique of carry propagation called "carry-save." This technique eliminates the need for completing the propagation of carries through the adder for each successive addition. At the completion of the multiplication all unpropagated carries are taken care of at once. Because the carry, or borrow, propagation is a significant part of the total addition time and there are up to 48 additions to perform, this method can be very effective.

Finally, it is possible to speed multiplication by operating on more than one bit of multiplier at a time. This method requires formation of multiples of the multiplicand. For example, if two multiplier bits are examined at one time, it is necessary to form the multiplicand and the two times and three times value. Then the correct multiple is added to the partial product depending on the value of the two multiplier bits being examined.

The 6600 Multiply Units use all three methods described above to provide very high-speed multiplication functions, as follows:

1. two halves of the multiplier are handled at once.
2. two bits of the multiplier are examined in each half.
3. carry-save adders are used.

SEQUENCE

Operands Xj and Xk arrive together at the input registers of the Multiply Unit. Immediately, operand Xk is shifted left one bit position and entered, along with the original Xk, in an adder to form 3Xk. See Figure 57.

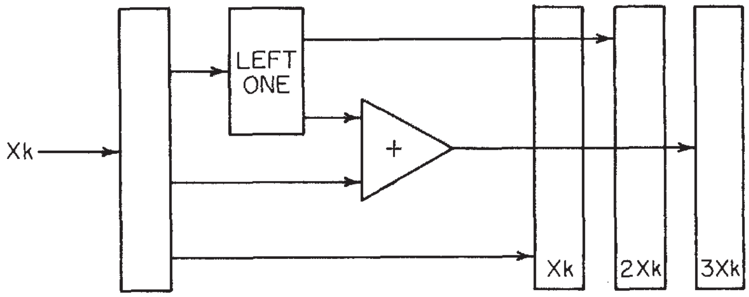


FIGURE 57

When these multiples of the multiplicand Xk are available, a sequence of four identical steps is taken utilizing the two halves of the multiplication. Each of the four steps accomplishes the multiplication of six bits in each half. This is diagrammed in Figure 58, showing three "carry-save" layers of network followed by a holding register.

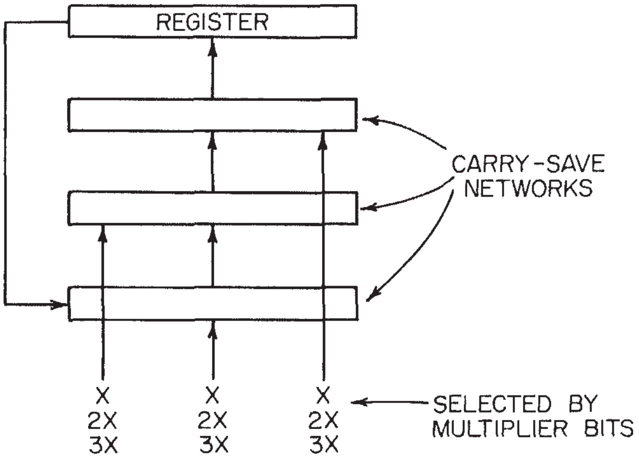


FIGURE 58

Each carry-save layer forms the sum of the partial product and the multiplicand "multiple" selected by two bits of the multiplier. The result of this sum is passed on to the next layer, shifted to the right two bit positions. The value placed in the holding register at the end of each step is the partial product of six bits of multiplier and the multiplicand and is shifted right by six places.

This value is returned to the first carry-save layer for the next identical step. At the end of four complete steps, there will exist in the two holding registers the two unmerged halves of the product.

The final product is formed by merging these two halves in a parallel adder and allowing all "saved" carries to fully propagate. On completion of this merge operation, the result is transmitted. In case the original operands were normalized, the result may be transmitted with a normalizing single place left shift, as needed.

CARRY-SAVE NETWORK

The carry-save network has an interesting and very convenient property. In this network, two numbers may be added in such a way as to produce two answers which fully specify the result but have a form suitable for temporarily holding carries. The network is actually a three-input, two-output adder. The two outputs, called pseudo-sum and pseudo-carry, can be applied to two of the three inputs in a subsequent step, the third input being a normal number. Carries generated in the repeated use of the network are never "lost."

Referring to previous derivations of the parallel adders in the Fixed Add Unit and Add Unit, a derivation of the three-input adder is given below.

It is assumed in this case that the adder network is "additive," a convenience arising from the fact that no subtraction is involved in the Multiply Unit and also that the operands are forced positive.

$$\text{Carry-Save Pseudo-sum } n = (An \oplus PSn) \oplus Cn$$

where An = net addend
PSn = pseudo-sum from previous addition
Cn = carry into bit n

$$Cn = A_{n-1}PS_{n-1} + PC_{n-1}$$

where PC_{n-1} = pseudo-carry from previous addition

$$\text{Carry-Save Pseudo-carry } n = (An \oplus PSn)Cn.$$

The effect of the above treatment of carry is seen to be a one-place carry propagation. In general, the carry is defined as:

$$Cn + 1 = AnBn + (An + Bn)Cn \qquad \text{or} \qquad Cn + 1 = AnBn + (An \oplus Bn)Cn$$

The pseudo-carry term is, in effect, a temporary storage of the second half of this general carry equation. Similarly, the pseudo-sum term is, in effect, a temporary storage of the sum accounting only for the carry generated one place to the right and the pseudo-carry from the previous step.

It is of some interest to test if carries generated in subsequent additions can break down this scheme.

For this test the potential conflict between the two terms making up the carry Cn are suspect. The two terms PSn and PCn, however, can be seen to be mutually exclusive. It is not possible to generate a carry term Cn with a value greater than one.

The following exercise may aid in explaining a simple addition of three numbers A, B, and C.

$$\begin{array}{lll} \text{A} & 000\ 000\ 101 & (005)_8 \\ \text{B} & 011\ 111\ 100 & (374)_8 \\ \text{C} & \underline{000\ 110\ 000} & \underline{(060)_8} \\ & 100\ 110\ 001 & (461)_8 \end{array}$$

$$\begin{array}{rl} \text{A} & 000\ 000\ 101 \\ +\text{B} & \underline{011\ 111\ 100} \\ & |\!\longleftarrow\!\!\!-\!\!\!-\!| \end{array}$$

Carry
Propagated

$$\begin{array}{rl} \text{SUM} & 100\ 000\ 001 \\ +\text{C} & \underline{000\ 110\ 000} \\ & 100\ 110\ 001 \end{array}$$

This shows a carry generated in the first sum of A and B propagating to the most significant bit. Following is the carry-save version of this example.

$$\begin{array}{rr} \text{A} & 000\ 000\ 101 \\ +\text{B} & \underline{011\ 111\ 100} \\ \text{Pseudo-Sum } PS_1 & 011\ 110\ 001 \\ \text{Pseudo-Carry } PC_1 & 0000\ 010\ 00 \end{array}$$

Note here that the pseudo-carry is shown shifted left one place, in position for the next addition. Note also that the carry generated has acted on the pseudo-sum just to the left and is temporarily stored as a pseudo-carry "passed on" to the left, therefore requiring further propagation.

Continuing the carry-save version:

$$\begin{array}{rr} PS_1 & 011\ 110\ 001 \\ PC_1 & 0000\ 010\ 00 \\ +\text{C} & \underline{000\ 110\ 000} \\ PS_2 & 010\ 110\ 001 \\ PC_2 & \underline{0\ 010\ 000\ 00} \\ \text{FINAL SUM} & 100\ 110\ 001 \end{array}$$

The carry-save mechanism is an extremely simple one, being made up of Exclusive OR circuits, OR and AND circuits. The DCTL circuit offers a very convenient Exclusive OR as shown in the carry-save adder of Figure 59.

An example of this convenience is the exclusive OR completing the solution of pseudo-sum, for simplicity described as $Zn \oplus Cn$. The pseudo-

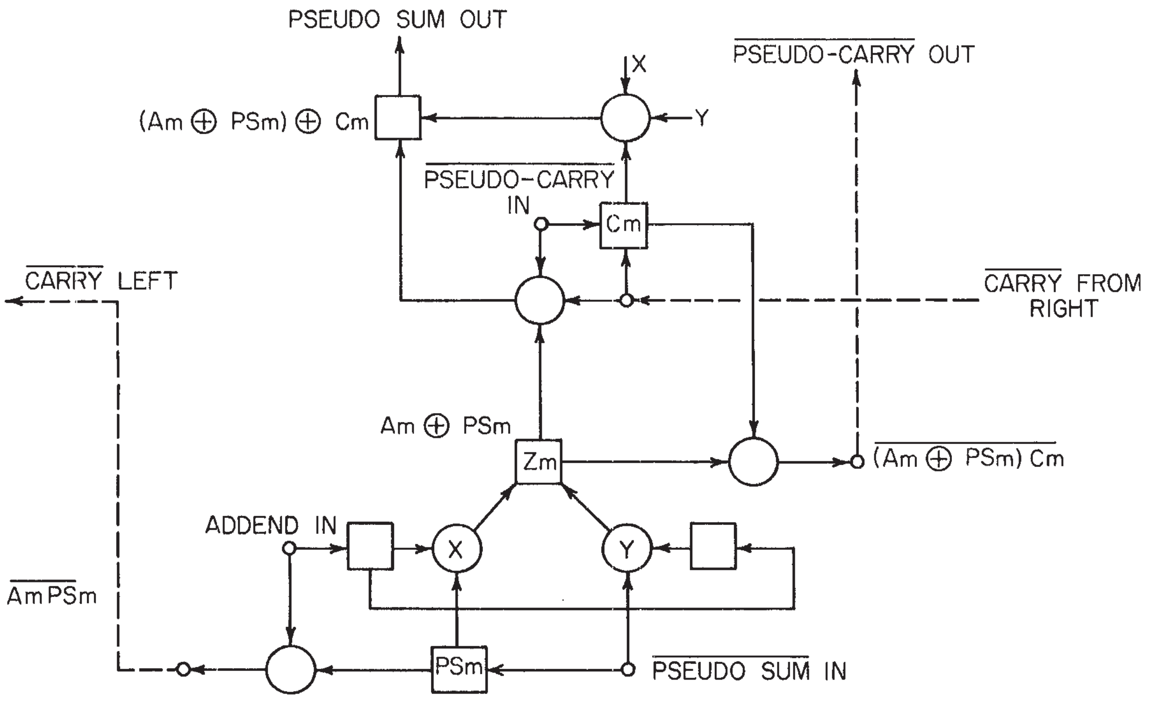


FIGURE 59

sum is formed using the "inputs" of Zn and Cn to obtain the NOT or complement of each. The purpose is both total hardware and time through the network, the second being most important. If, for example, an additional term were included for $\overline{Zn}$, this additional inverter time would be added to the network. The maximum number of inverters through the network is five.

Three such networks can be connected, with appropriate shifts wired in, such that only fifteen inverters are involved in the long path. This can include the register, utilizing the clear-set technique for setting the flip-flop. The result of this configuration is three carry-save additions in one minor cycle of 100 nanoseconds, the same time needed for a single conventional parallel addition. This is especially useful for repetitive additions as in multiplication.

These three carry-save networks are located on logic modules in a manner consistent with ground rules of loading and pin limits. The block diagram of Figure 60 shows this layout on three adjacent logic modules.

The addend input to the carry-save network in the previous discussion is now identified as Mn, the selected Multiplicand multiple. Control over the selection of which multiple at each "layer" is shown at the left of the Figure. For example, multiplier bits 0 and 1 are translated to select M, the multiplicand, 2M or 3M, or, of course, zero. The bits of M are entered as shown. Each layer of this network is required to accomplish a shift to the right of the multiplier and partial product relative to the multiplicand. In this case, it is convenient to hold the partial product and shift the multiplicand left at each layer. The shift is two bit positions since two bits of multiplication are accomplished in each layer. For purposes of clarity the

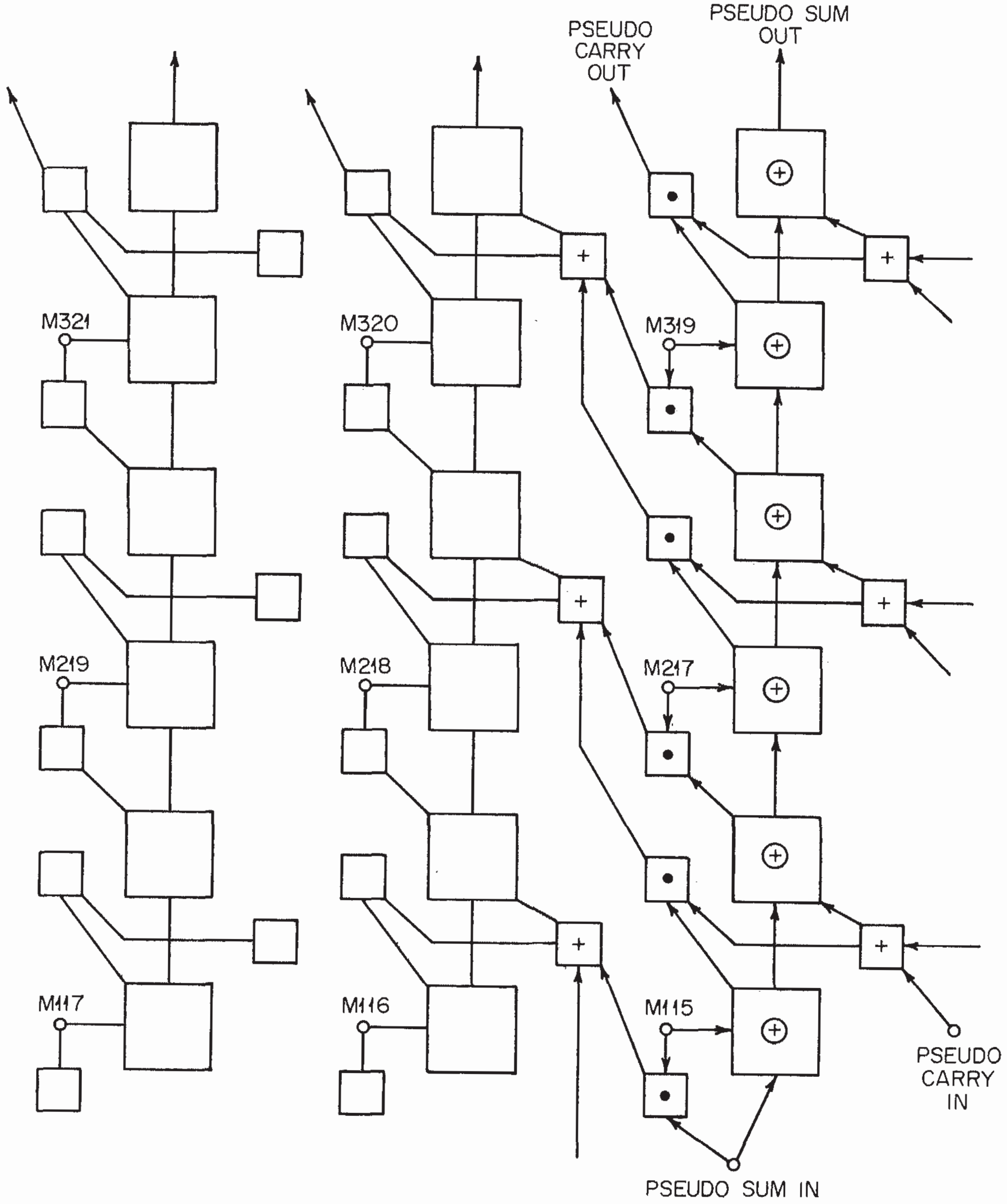

PSEUDO CARRY OUT
PSEUDO SUM OUT
M321
M320
M319
M219
M218
M217
M117
M116
M115
PSEUDO CARRY IN
PSEUDO SUM IN

FIGURE 60

multiplicand terms are defined as M117, M217, and M317, showing the independence of each layer in terms of multiple selection but retaining the bit identification. These three are all bit 17 of the selected multiple.

The result of the total network for one half of the multiply is shown in Figure 61.

Fifty bits of basic network are required to allow addition of the 3X multiple of the 48-bit multiplicand. At each layer, two bits of product are removed to a holding register, a result of the right shift of partial product. After four iterations through this network, each half of multiplier has produced 24 bits "off the end" and 48 bits of remaining partial product. Since

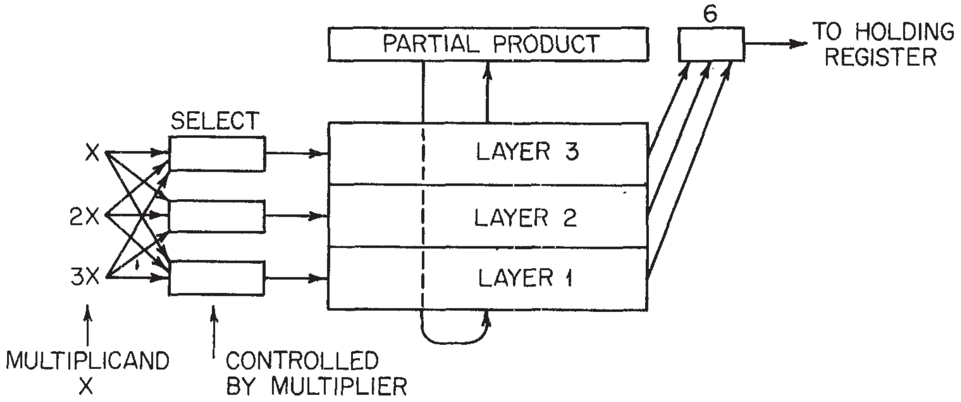


FIGURE 61

one-half of the multiplier is 24 bits in length, the maximum value positive number contained is $2^{24} - 1$. Thus the maximum value in each half of partial product is as follows:

$$(2^{24} - 1)(2^{47} - 1) = 2^{71} - 2^{47} - 2^{24} + 1.$$

This requires 72 bits of register.

The "upper half" partial product overlaps the "lower half" as follows, with letters X indicating octal digits directly from each network and letters Y indicating "off the end" digits.

```
upper    XXXX XXXX XXXX XXXX YYYY YYYY
lower              XXXX XXXX XXXX XXXX YYYY YYYY
```

The merging addition requires an adder with 72 bits of length. Since the lower Y terms do not influence this merging addition, the pseudo-sum and pseudo-carry terms, making up this portion of the partial product, can be summed. For convenience, these are summed six bits at a time during the four multiply iteration cycles.

MERGE

At the completion of the four iteration cycles there are four numbers appearing as outputs of the carry-save networks. These are the pseudo-sum and pseudo-carry for each half. A minor modification to the basic carry-save network can be used to convert it into a full adder. This modification plus the addition of a small amount of carry propagation network is shown in Figure 62 (page 98).

The output of the carry network is shown brought into the input normally used by the multiplicand. The time for complete carry propagation is approximately one minor cycle.

The above modification is made to the top layer, leaving the lower two layers untouched. The four values to be merged can then take advantage of these lower layers. For this purpose, the "lower" half carry-save network is used for the merge. The pseudo-sum from the "upper" partial product is

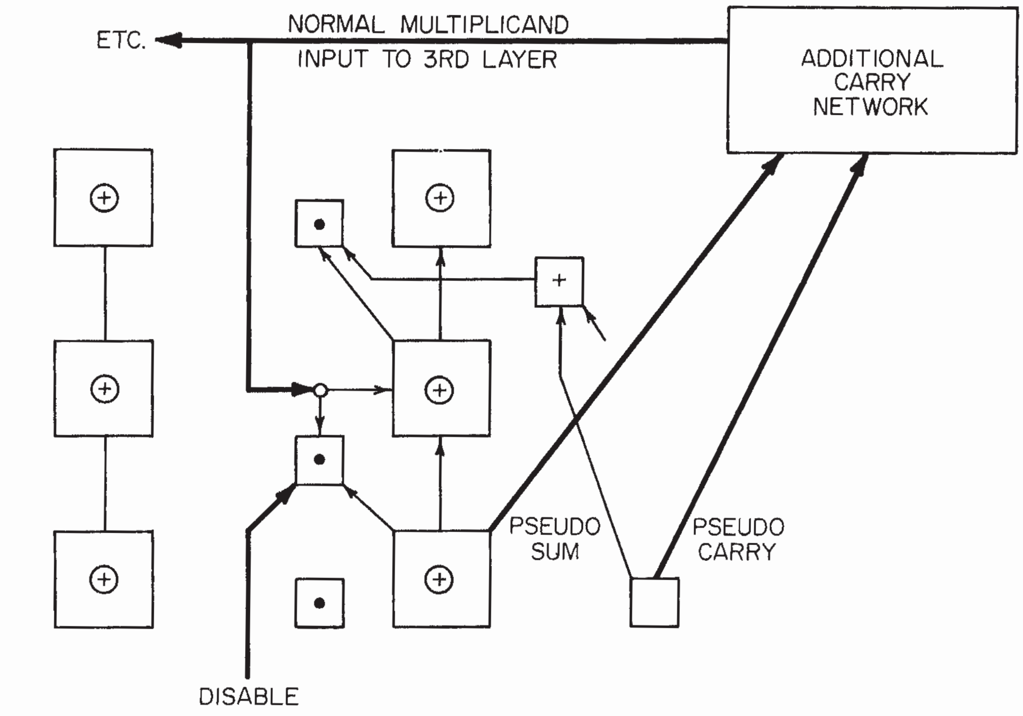

ETC.
NORMAL MULTIPLICAND
INPUT TO 3RD LAYER
ADDITIONAL
CARRY
NETWORK
PSEUDO
SUM
PSEUDO
CARRY
DISABLE

FIGURE 62

brought into the input normally used by the multiplicand for the bottom layer. Similarly, the pseudo-carry from the "upper" half partial product is brought into the middle layer. See Figure 63. The carry from the lower 24 bits of product is introduced into the long carry network as shown.

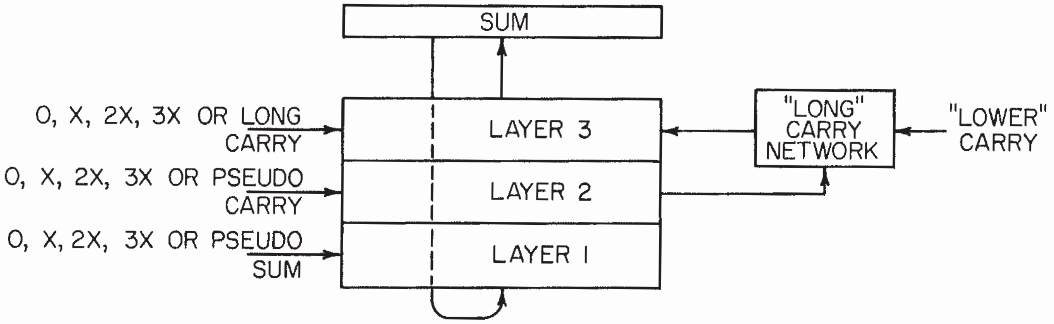

SUM
0, X, 2X, 3X OR LONG CARRY
0, X, 2X, 3X OR PSEUDO CARRY
0, X, 2X, 3X OR PSEUDO SUM
LAYER 3
LAYER 2
LAYER 1
"LONG" CARRY NETWORK
"LOWER" CARRY

FIGURE 63

The lower half network is, of course, too small to complete the 96 bits of final product. An additional adder for these extra bits is included.

EXPONENT

Each Multiply Unit contains logic for the exponent calculation needed. Two additions and one subtraction are required of this logic, as follows.

1. Sum of Xj exponent and Xk exponent used for the double precision, or lower, product.
2. Sum of 1 above and $48_{10}(60_8)$ used for the single precision, or upper, product.
3. Difference of 1. or 2. above and one used when single place left shift is needed to normalize result.

Figure 64 on page 100 is descriptive of this logic.

Operands Xj and Xk are forced positive on entry. This places the respective exponents of Xj and Xk in the input registers in true form. Since these input registers are used by both Multiply Units, they are emptied within one minor cycle. The two sums described above are formed, and one is selected by the function currently being executed in the unit. The decrement by one is performed and held until the final coefficient product is completed. As described previously, the final result will be normalized if the input operands were normal. This means a left shift of one place for the coefficient and a reduction by one of the final exponent. The test is made near the end of the multiply sequence. The proper one out of four possibilities is picked for the coefficient result, together with the proper one out of two exponents. The results are complemented if the original signs were unlike. The four possible paths for the coefficient are:

- Direct output upper
- Direct output lower
- Left shift output upper
- Left shift output lower

Tests of the end-case conditions for the exponents are also made. It is possible to generate exponent overflow, infinity, indefinite, and underflow in this unit.

ROUND

Rounding is provided in each Multiply Unit for the Single Precision, or upper, result. Determination of the round condition is made on the lower product coefficient. Simply stated, if the lower coefficient of the product is equal to or greater than one half the upper, a round-off is required.

Truncation of the 96-bit product coefficient has the effect of rounding downward or toward zero. This is an attribute of the one's complement number representation. Of course, because both operands are initially forced positive, it would also hold true of two's complement representation as well.

A common but time-consuming method of rounding is the addition of the number one-half to the upper product integer, and allow carries to propagate into the upper product. The effect is to increase the upper product by one for lower half products equal to, or greater than, one-half relative to the upper. This kind of rounding requires a full addition time.

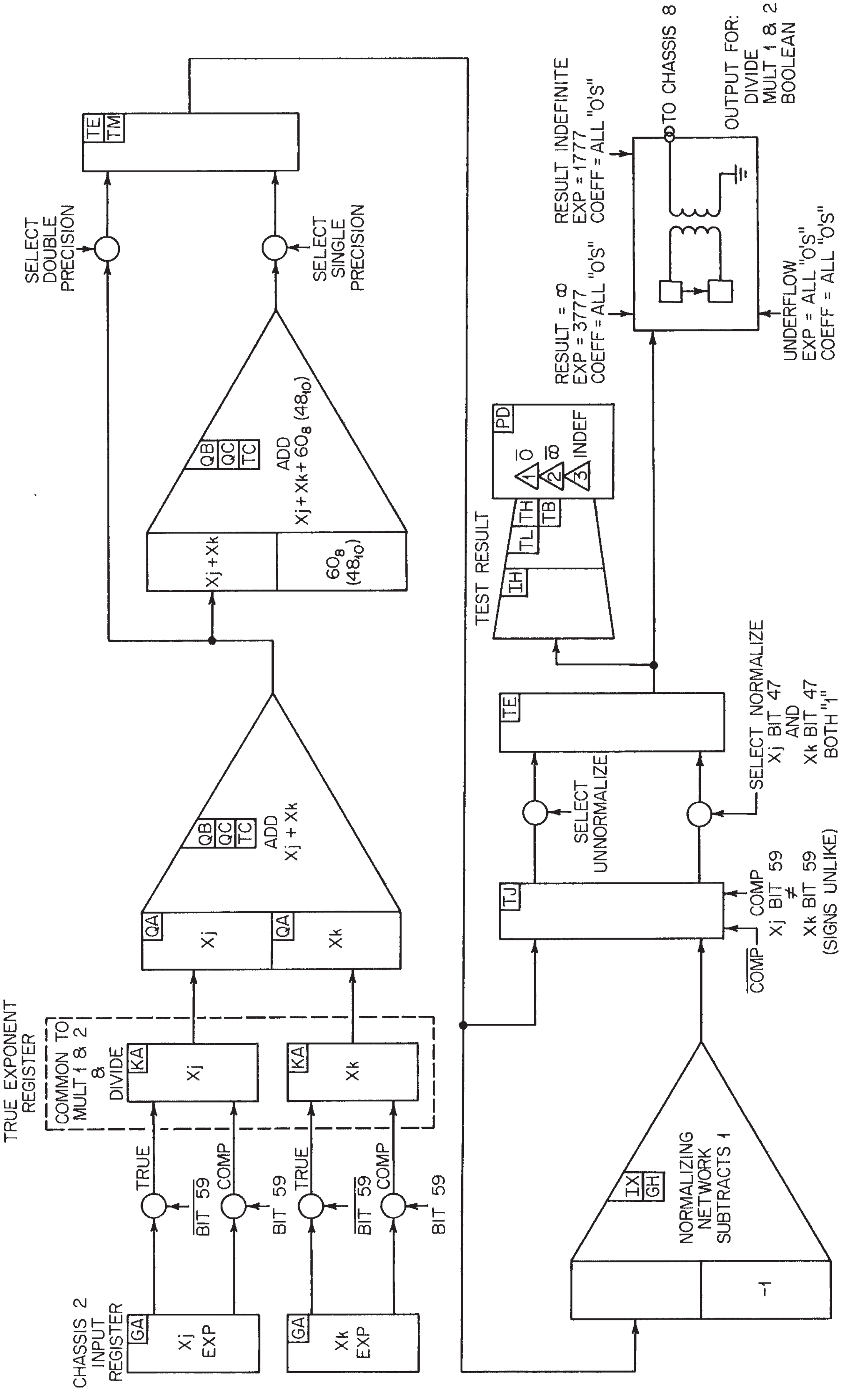

CHASSIS 2 INPUT REGISTER
GA
Xj EXP
GA
Xk EXP
BIT 59
BIT 59
TRUE
COMP
BIT 59
BIT 59
TRUE
COMP
TRUE EXPONENT REGISTER
COMMON TO MULT 1 & 2 & DIVIDE
KA
Xj
KA
Xk
QA
Xj
QA
Xk
QB
QC
TC
ADD Xj + Xk
Xj + Xk
60_8 (48_{10})
QB
QC
TC
ADD Xj + Xk + 60_8 (48_{10})
SELECT DOUBLE PRECISION
SELECT SINGLE PRECISION
TE
TM
NORMALIZING NETWORK SUBTRACTS 1
IX
GH
−1
TJ
$\overline{\text{COMP}}$
COMP
Xj BIT 59 ≠ Xk BIT 59 (SIGNS UNLIKE)
SELECT UNNORMALIZE
SELECT NORMALIZE Xj BIT 47 AND Xk BIT 47 BOTH "1"
TE
TEST RESULT
IH
TL
TH
TB
PD
1 $\overline{0}$
2 $\overline{\infty}$
3 INDEF
RESULT = ∞
EXP = 3777
COEFF = ALL "0'S"
RESULT INDEFINITE
EXP = 1777
COEFF = ALL "0'S"
UNDERFLOW
EXP = ALL "0'S"
COEFF = ALL "0'S"
TO CHASSIS 8
TO CHASSIS 8
OUTPUT FOR:
DIVIDE
MULT 1 & 2
BOOLEAN

FIGURE 64

Rounding in the Multiply Unit is accomplished without additional time. The technique used is to preset the adder network such that the preset value will be added to the upper product. It is assumed that rounding is desired primarily in conjunction with normalized numbers. Approximately half the products of normalized numbers will require a single-place left shift to normalize the result. A preset value for round receives the effect of such a shift. Therefore, the preset value chosen is the number one-fourth relative to the integer upper product. As a result, the following round conditions apply.

1. Half of all products which require a left shift will be rounded up by one. The other half will round down by truncation.
2. One quarter of all products which require no left shift will be rounded up by one. The other three quarters will round down by truncation.

The net effect of this pre-round technique is to bias the round slightly toward zero. This deviation is considered satisfactory and reasonable because of the 48-bit coefficient length and the ability to perform double-precision multiplication.

G. DIVIDE UNIT

The slowest of the Central Processor functional units is the Divide Unit. Floating point division requires 2900 nanoseconds, while Population Count, a function also assigned to this unit, is accomplished in 800 nanoseconds. The three instructions executed in the Divide Unit are:

44 Floating Divide Xj by Xk to Xi,

45 Round Floating Divide Xj by Xk to Xi,

47 Count the number of ones in Xk to Xi,

In the design of this unit, one humorous incident stands out and should be related. The instruction codes shown above represent a very simple and convenient combination if code 46 is included. However, code 46 was selected as the PASS instruction. As the reader has perhaps already suspected, the PASS instruction design ended up triggering a complete Divide Sequence! Needless to say, this minor embarrassment was corrected.

There are very few really effective strategies available for the design of divide logic. The basic method operates much like the pencil and paper method; the successive subtraction of the divisor from the dividend followed by a left shift of the dividend and quotient relative to the divisor.

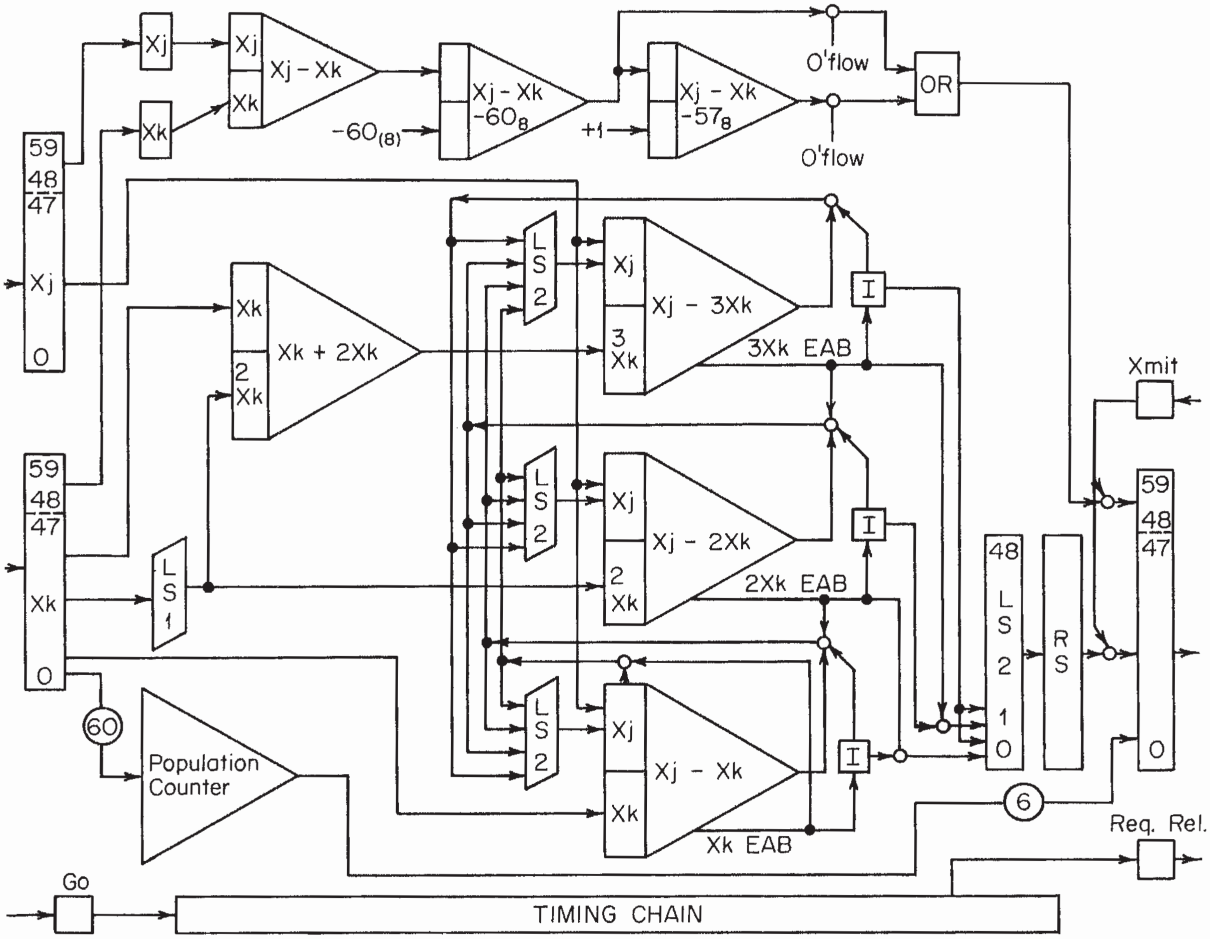
The following example illustrates the method using four octal digits to represent the divisor coefficient and the dividend coefficient.

Xj = 7604
Xk = 5213

```
                       1.3613
Xj/Xk = 5213        7604.0000
                    5213
                    2371 0
                    1764 1
                     404 70
                     375 02
                       7 660
                       5 213
                       2 4450
                       1 7641
                         4607
```

While an octal method could be used, the Divide Unit instead forms only two bits of Quotient in one step. To do even this, three subtraction networks are required as shown in Figure 65.

The operands are initially forced "positive." Three values are simultaneously subtracted from the partial dividend. These values are the divisor



Xj
Xk
Xj − Xk
−60(8)
Xj − Xk −60₈
+1
Xj − Xk −57₈
O'flow
O'flow
OR
59
48
47
Xj
0
Xk
Xk + 2Xk
2 Xk
L S 2
Xj − 3Xk
3 Xk
3Xk EAB
I
Xmit
Xj − 2Xk
2 Xk
2Xk EAB
L S 1
Xk
0
59
48
47
48 L S 2 1 0
R S
Xj − Xk
Xk EAB
60
Population Counter
6
Req. Rel.
Go
TIMING CHAIN

FIGURE 65

and its second and third multiple, a design condition rather similar to the Multiply Unit. Because the operands are initially forced positive, the dividend is positive and the divisor is negative in preparation for the subtraction.

The Divide Unit design assumes that normalized arithmetic is being used or that other program techniques are applied to protect against divide overflow. One technique is to normalize at least the divisor. Methods of unnormalized arithmetic, such as significance arithmetic, can also be used. In any event, the unit corrects only for a single-bit overflow such as the case cited above.

Following this example through, multiples of the divisor are first formed.

Divisor	Dividend
Xk = 5213 (Octal)	Xj = 7604
2Xk = 12426	
3Xk = 17641	

All three multiples of the divisor are simultaneously subtracted from the dividend. The largest of the three which can successfully be subtracted without changing the sign of the resultant partial dividend will define the quotient bits for this iteration. The first iteration defines a quotient of 01.XXX . . . X binary since only Xk can be subtracted. The resultant partial dividend is also picked from the first subtractor and entered into all three registers associated with the three subtractor networks.

For the second iteration, the partial dividend is shifted left two bit positions. This is shown as follows.

Dividend	7604	Octal
Divisor 1X	$\underline{-5213}$	
	2371	Shifted left 2 bits = 11744

Trial subtraction for the second iteration again finds the largest acceptable subtraction of Xk. The following table is a consolidation of all seven iterations showing the partial dividend after the successful subtraction, unshifted and shifted, as well as the quotient.

Step	Partial Div. Shifted	Partial Div. Unshifted	Quotient (Binary)
1	7604	2371	$\underline{0.1}$.XXXXXXXXXXXX
2	11744	4531	01.$\underline{01}$XXXXXXXXXX
3	22544	2703	01.01$\underline{11}$XXXXXXXX
4	13414	766	01.0111$\underline{10}$XXXXXX
5	3730	3730	01.011110$\underline{00}$XXXX
6	17540	5112	01.01111000$\underline{10}$XX
7	24450	4607	01.0111100010$\underline{11}$

The quotient is therefore 1.3613 octal which is a single-place overflow. This is corrected by a single right shift, producing the coefficient result .5705 octal. The exponent of the result is increased by one in conjunction with this shift.

Because coefficients are really 48 bits long, not 12, the above procedure requires twenty-five subtractions and twenty-four shifts. Each subtraction takes one minor cycle since the borrows generated during each subtraction must propagate to the sign bit in order to determine the success of each subtraction.

Because the signs of the numbers are known, it is a considerable, in fact, critical advantage to look at the end-around-borrow, EAB, rather than the sign. With a 48-bit subtract network, EAB can be determined in fewer inverters than the sign. To pick the correct network output requires combining the EAB signals from all three networks as shown in Figure 66.

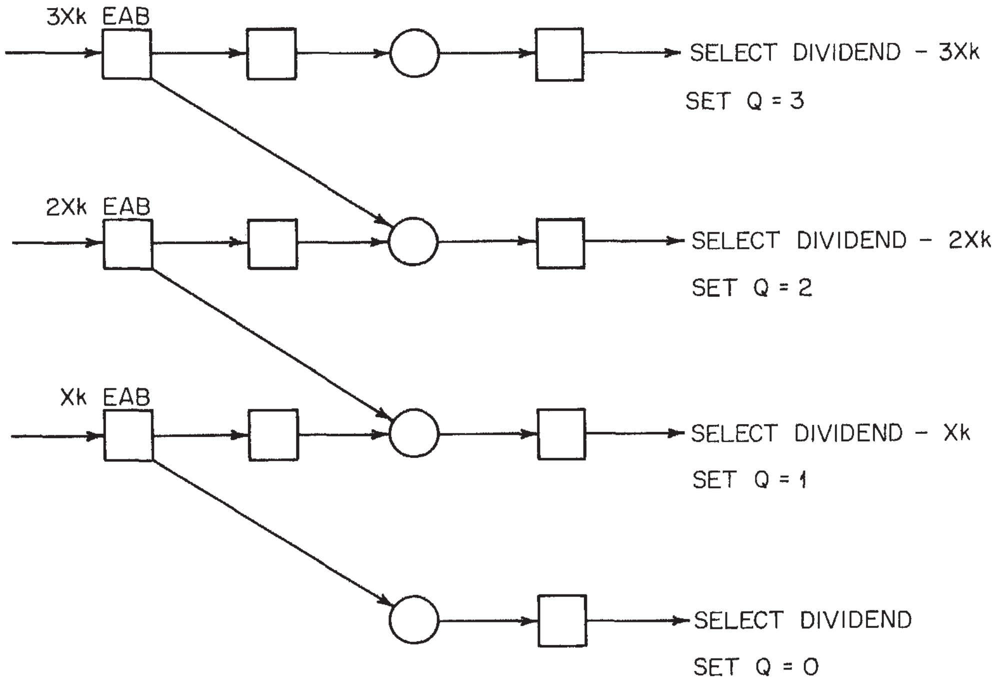


FIGURE 66

This logic is a direct result of the relationship of end-around-borrow to the relative sizes of the partial dividends and the particular subtractor. There will be an EAB *only* if the applicable divisor multiple is *greater than* the dividend.

EXPONENT

A series of add networks are connected in a manner similar to the Multiply Unit in order to perform the exponent calculation. The true exponents are extracted from the original operands Xj and Xk. The exponent of the divisor Xk is subtracted from the exponent of the dividend Xj.

Because the result of the division is essentially a fraction as shown in the example, a reduction in the exponent is necessary to conform to the correct positioning of the binary point. This reduction of 48_{10}, or 60 octal, is accomplished by a second add network. Finally, the overflow correction requires an increase of the resultant exponent by one.

ROUND

To counteract the effect of truncation of the quotient, the Divide Unit includes the ability to round-off. In this Unit, as in Multiply and Add Units, the round is achieved with no increase in time. A round value is preset into the dividend in order to effect an increase of the final quotient by one-half. As in the other units, the overflow and normalize adjustments tend to affect this type of round.

In the Divide Unit, about fifty per cent of the quotients resulting from using normalized dividend and divisor require the overflow right shift adjustment. For the half requiring no shift adjustment, the preset round value of one-third relative to the integer dividend has several interesting properties. This value, if essentially added to the original dividend, is also divided by the divisor. If the divisor is normalized, the possible range is $1/2{\cdot}2^n$ to just less than 2^n. The effect on the round value is to give it a range from one-third to two-thirds, on completion of the division. For the no-shift cases, this centers around the desired value one-half. Again, as in Multiply, the right shift cases tend to bias the round-off slightly toward zero.

POPULATION COUNT

The count of the number of ones in a sixty-bit word is accomplished in the Divide Unit in 800 nanoseconds. The logic of this "population" counter is essentially a tree of adders, as can be seen in Figure 67 (page 106).

A first column of circuits is constructed to form three-bit quantities of the number of ones in four-bit groups. There are, as a result, fifteen octal quantities to be added. Taken two at a time, four add cycles are needed. The logic of a single four-bit group converted to a three-bit binary number is an interesting combination in Figure 68 (pages 108–109).

The four bits generate two Exclusive OR terms which combine in another Exclusive OR to establish the least significant bit of the count value. The remainder of the logic in the figure is self-explanatory.

H. INCREMENT UNITS

Two Increment functional units are included in the Central Processor. Each unit is capable of performing fixed point addition and subtraction on 18-bit fixed point numbers. These operations are needed in:

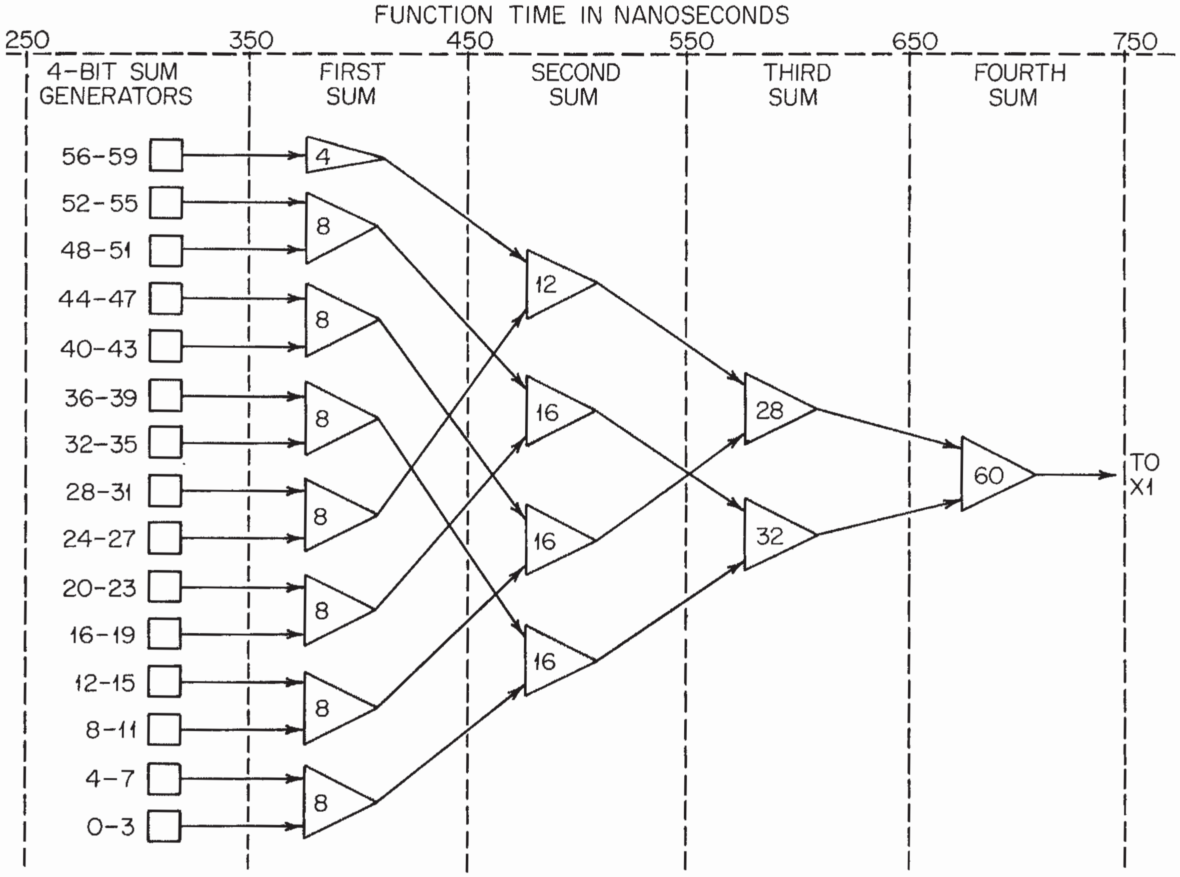

FUNCTION TIME IN NANOSECONDS
250
350
450
550
650
750
4-BIT SUM GENERATORS
FIRST SUM
SECOND SUM
THIRD SUM
FOURTH SUM
56-59
52-55
48-51
44-47
40-43
36-39
32-35
28-31
24-27
20-23
16-19
12-15
8-11
4-7
0-3
4
8
8
8
8
8
8
8
12
16
16
16
28
32
60
TO X1

FIGURE 67

- Indexing
- Reading and storing arithmetic operands
- Conditional branch tests.

A set of instructions in the Central Processor is devoted to preparing, incrementing, and modifying the addresses for reading and storing operands. The effect of these instructions is to generate a new address in one of the eight A registers and, at the same time, to initiate a storage reference at the new address. These instructions are listed below.

50	Sum of Aj and K to Ak	30 bits
51	Sum of Bj and K to Ai	30 bits
52	Sum of Xj and K to Ai	30 bits
53	Sum of Xj and Bk to Ai	15 bits
54	Sum of Aj and Bk to Ai	15 bits
55	Difference of Aj and Bk to Ai	15 bits
56	Sum of Bj and Bk to Ai	15 bits
57	Difference of Bj and Bk to Ai	15 bits

Note that the first three are the long format of thirty bits in order to describe an eighteen-bit quantity K. Shown in Figure 5 of Chapter II is the relationship of the A registers to the X registers. Specific read trunks are provided from central storage to registers X1 through X5. Specific store trunks are provided from registers X6 and X7 to central storage. Whenever a result of one of the above instructions enters one of these seven A registers, a storage reference is initiated causing a read or a store to or from the specific "partner" X register. Registers A0 and X0 do not participate in central storage operations but are reserved for Extended Core Storage usage.

Another set of instructions is devoted to fixed point calculation for indexing, manipulation of constants, and assorted other uses. These are grouped such as to produce changes in the B Increment registers and is the X operand registers, as follows.

60	Sum of Aj and K to Bi
61	Sum of Bj and K to Bi
62	Sum of Xj and K to Bi
63	Sum of Xj and Bk to Bi
64	Sum of Aj and Bk to Bi
65	Difference of Aj and Bk to Bi
66	Sum of Bj and Bk to Bi
67	Difference of Bj and Bk to Bi

70	Sum of Aj and K to Xi
71	Sum of Bj and K to Xi
72	Sum of Xj and K to Xi
73	Sum of Xj and Bk to Xi
74	Sum of Aj and Bk to Xi
75	Difference of Aj and Bk to Xi
76	Sum of Bj and Bk to Xi
77	Difference of Bj and Bk to Xi

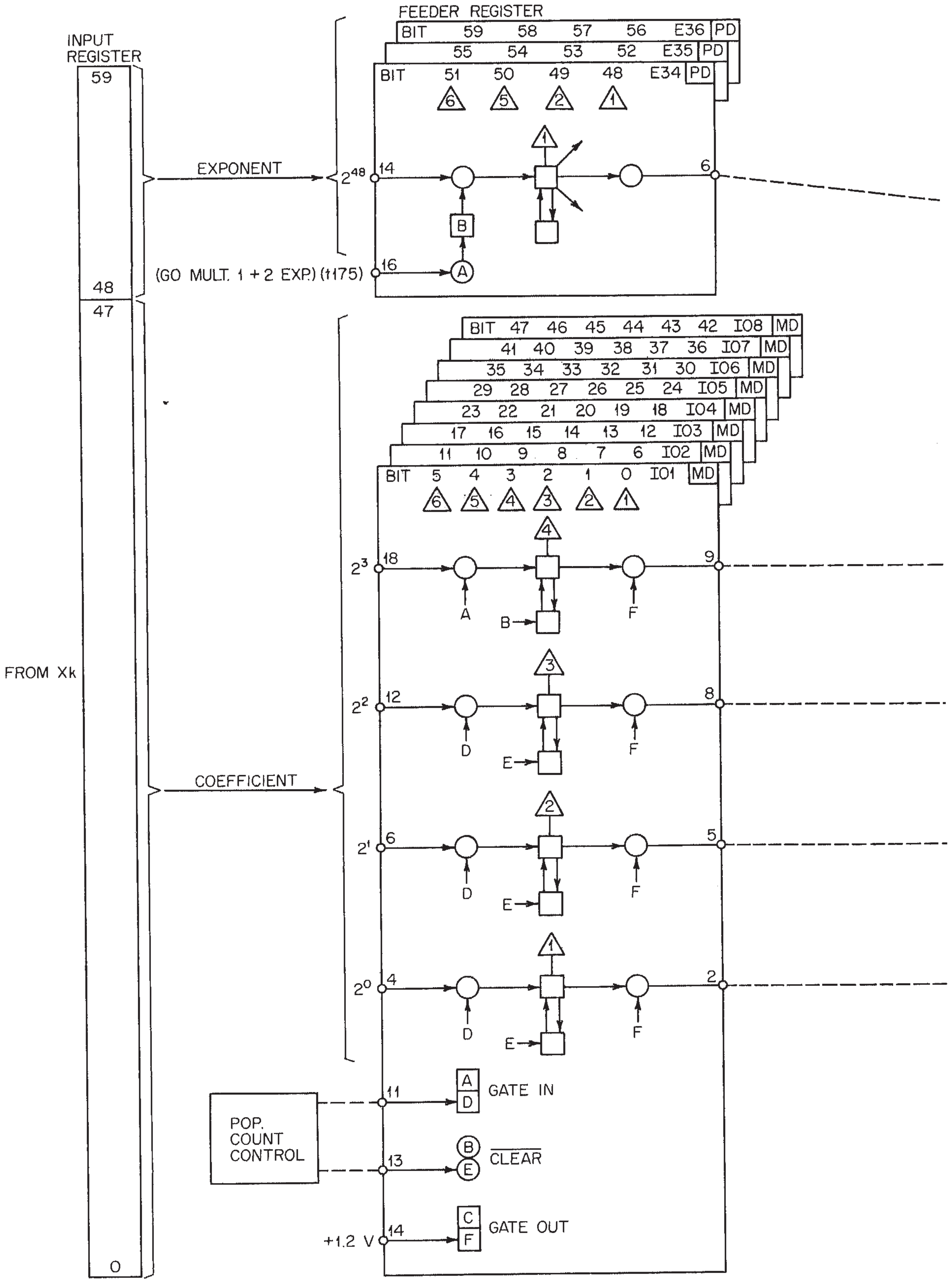
Note that the three sets of instructions differ only in the specification of the result register set X, B, or A. Although many more combinations of addition and subtraction are possible on the registers, only the foregoing were implemented in the design and are considered to be the most useful.

The Increment Units are also utilized as "partner" units to the Branch Unit for conditional jumps, as described in the following instructions. Again, as in the Fixed Add Unit, the Increment Unit is begun simultaneously with the Branch Unit, as a "partner." The Increment Unit accomplishes the conditional tests while the Branch Unit is preparing the destination address.

02	Go to K + Bi	30 bits

04	Go to K if Bi $=$ Bj	30 bits
05	Go to K if Bi $\neq$ Bj	30 bits
06	Go to K if Bi $\geq$ Bj	30 bits
07	Go to K if Bi $<$ Bj	30 bits

The first jump instruction above, 02, is an unconditional jump to location K plus the contents of register Bi. In this case, the Increment Unit performs the 18-bit addition needed to determine the jump destination.


FEEDER REGISTER
BIT 59 58 57 56 E36 PD
55 54 53 52 E35 PD
BIT 51 50 49 48 E34 PD
6 5 2 1
4
INPUT REGISTER
59
48
47
0
EXPONENT
2^{48}
14
6
B
16
A
(GO MULT. 1 + 2 EXP.) (t175)
BIT 47 46 45 44 43 42 I08 MD
41 40 39 38 37 36 I07 MD
35 34 33 32 31 30 I06 MD
29 28 27 26 25 24 I05 MD
23 22 21 20 19 18 I04 MD
17 16 15 14 13 12 I03 MD
11 10 9 8 7 6 I02 MD
BIT 5 4 3 2 1 0 I01 MD
6 5 4 3 2 1
4
2^3
18
9
A
B
F
3
2^2
12
8
D
E
F
FROM Xk
COEFFICIENT
2
2^1
6
5
D
E
F
1
2^0
4
2
D
E
F
POP. COUNT CONTROL
11
A
D
GATE IN
13
B
E
CLEAR
+1.2 V
14
C
F
GATE OUT

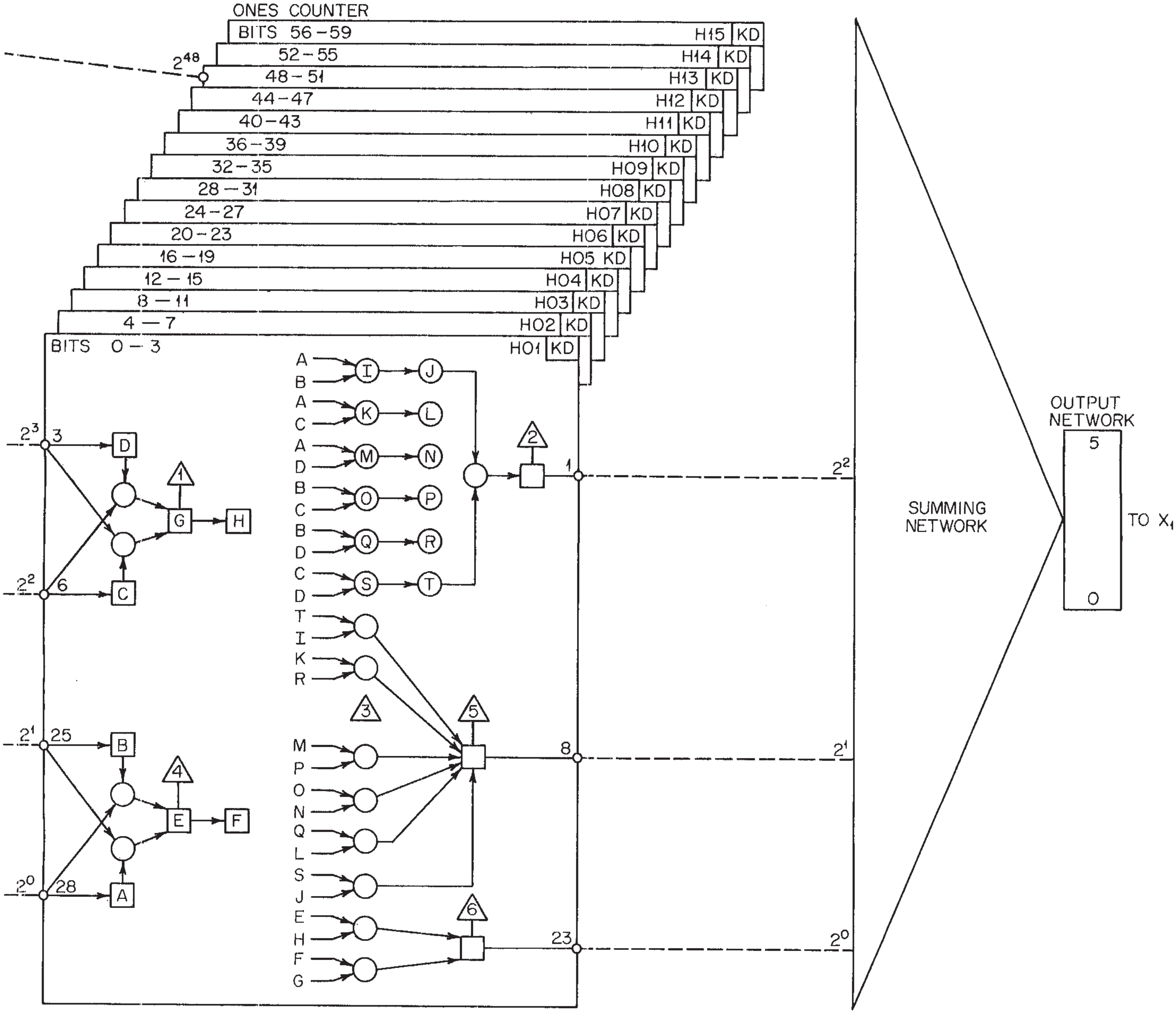

ONES COUNTER
BITS 56-59
52-55
48-51
44-47
40-43
36-39
32-35
28-31
24-27
20-23
16-19
12-15
8-11
4-7
BITS 0-3
H15 KD
H14 KD
H13 KD
H12 KD
H11 KD
H10 KD
H09 KD
H08 KD
H07 KD
H06 KD
H05 KD
H04 KD
H03 KD
H02 KD
H01 KD
2^48
2^3 3
2^2 6
2^1 25
2^0 28
1
8
23
SUMMING NETWORK
OUTPUT NETWORK
5
0
TO X1

FIGURE 68

The other four instructions, 04 through 07, are conditional jumps in which the Increment Unit performs the test on the specified B registers. In these five cases, the Increment Unit selected does not return a result to the registers. The result is transferred instead to the Control System of the Central Processor, as will be described in the following section on the Branch Unit.

The Increment Units are shown in Figure 69. The two Increment Units share, for convenience, a common add network. The identity of each unit is found in the result register. These units retain the result in temporary storage in the result registers until called for by the control system.

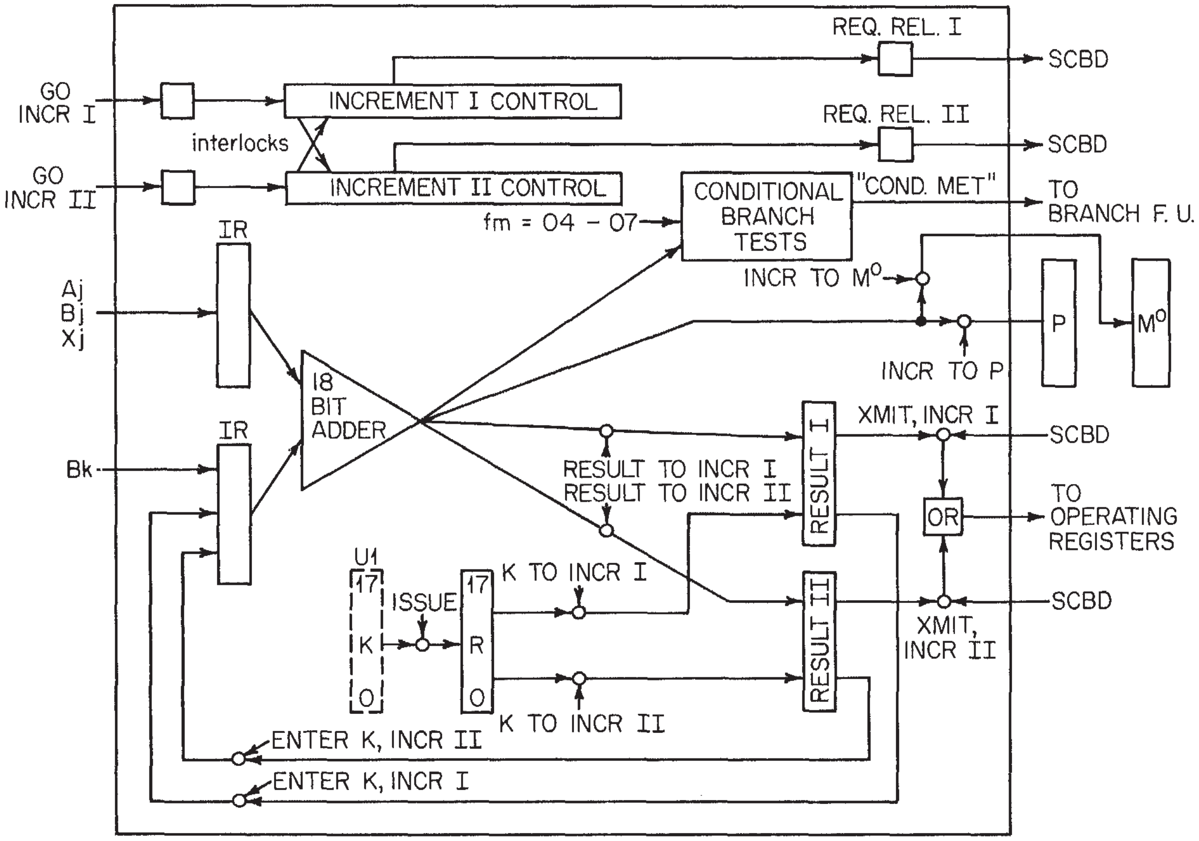

REQ. REL. I
SCBD
GO INCR I
INCREMENT I CONTROL
REQ. REL. II
SCBD
interlocks
GO INCR II
INCREMENT II CONTROL
CONDITIONAL BRANCH TESTS
"COND. MET"
TO BRANCH F. U.
fm = 04 - 07
IR
INCR TO M^0
Aj
Bj
Xj
P
M^0
18 BIT ADDER
INCR TO P
IR
XMIT, INCR I
SCBD
Bk
RESULT I
RESULT TO INCR I
RESULT TO INCR II
OR
TO OPERATING REGISTERS
U1
17
ISSUE
K
O
17
R
O
K TO INCR I
RESULT II
SCBD
XMIT, INCR II
K TO INCR II
ENTER K, INCR II
ENTER K, INCR I

FIGURE 69 Increment functional units—block diagram.

In those cases in which the K field from the instruction is used as one of the operands, this quantity is temporarily loaded into the appropriate result register. As will be described later, the other operand is brought into the unit, and the K entry is returned to the input at the same time.

STORAGE REFERENCES

In the set of instructions which specify a new address in one of the A address registers, there are two outputs of the selected Increment Unit. One of these outputs is connected to the specified A register, just as a normal functional unit output. The other is taken directly to the Central Storage Stunt Box. This technique obviously reduces the storage access time for a storage reference. A somewhat simpler form of design could have been to wait for the new address to be sent to the appropriate Address register and then to the Storage Stunt Box. This would have increased access time by two or more minor cycles.

The "computed GO TO" jump, 02, also requires the result of the Increment Unit add network. In this case, the destination value is sent to the Central Storage Stunt Box and to the Program Address Register P.

In some circumstances, the Increment adder network must hold the results temporarily because of a "backlog" of references entering the Central Storage Stunt Box. This is a rare event; thus the design tends to favor the "no-conflict" smooth-flow case. This portion of the control system is complicated, however, by a focusing of the Branch testing, indexing, and storage address manipulation activities in the Increment Units.

INCREMENT ADDITION

The fixed point numbers held in the A address registers and B increment registers are represented in one's complement form. Addition is accomplished in much the same manner as in the Fixed Add Unit and in the Add Unit. Operands entering the Increment Units are left in original form, regardless of sign.

In the case of operands from the X operand registers, which are of sixty-bit length, the sign is taken to be the 18th bit from the low or right end. Results going to the X registers from the Increment Unit produce sign extension in the selected X register.

I. BRANCH UNIT

The Branch Unit is unique in several ways relative to the other functional units.

- It uses "partner" units for conditional tests.
- It halts any further instruction issuing until the Branch is complete.
- The only result of the unit is a possible change in Program Address Register P, and a new program path, or a PASS.

The following instructions are executed by the Branch Unit.

010	Return Jump to K	
02	Go to K plus Bi	PARTNER-INCREMENT UNIT
030	Go to K if Xj = 0	PARTNER-FIXED ADD UNIT
031	Go to K if Xj ≠ 0	
032	Go to K if Xj is Positive	
033	Go to K if Xj is Negative	
034	Go to K if Xj is In Range	
035	Go to K if Xj is Out of Range	
036	Go to K if Xj is Definite	
037	Go to K if Xj is Indefinite	
04	Go to K if Bi = Bj	PARTNER-INCREMENT UNIT
05	Go to K if Bi ≠ Bj	
06	Go to K if Bi ≥ Bj	
07	Go to K if Bi < Bj	

In executing these Branch instructions, four major steps are taken.

Step 1 Determination of the condition specified in the conditional Branch instructions. This step is accomplished in the Partner Unit.
Step 2 Calculating the jump address.
Step 3 Determination of jumps to the Instruction Stack (Chapter VI).
Step 4 Initiating Storage reference for new instruction if not in the Instruction Stack.

PARTNER UNIT

In the conditional Branch instructions and in the 02 instruction, the Partner Unit must be free to proceed before the instruction can begin. Step 1, as described above, is executed at the same time as Step 2. This allows the condition test and the jump address calculation to be completed in the minimum time. If the condition is met, the new program address is transferred to the program address register P. If the condition is not met, the program continues to the next instruction.

INSTRUCTION STACK

Only the conditional Branch instructions, 03 through 07, are allowed to jump within the instruction stack. Eight words may be held in the instruction stack. These are loaded one at a time during the normal course of instruction fetch. Each new instruction word is entered at the bottom causing the older words to move up. Whenever a Branch instruction causes a jump out of the stack, all of the instruction words accumulated in the stack are declared void. New instruction words are then brought in as described in Chapter VI.

Two values are maintained in the control system for the stack, a depth

(D) and locator (L). The depth (D) is a measure of the valid instructions in the stack. This means that the value of D is set to zero after any Branch out of the stack and is increased by one for every new instruction word brought in. When the stack is full, D remains equal to seven. The locator L is used to specify the location in the stack of the instruction word currently in use. These values are all that are necessary to allow Step 3 above to be accomplished. The test is made in the following manner.

1. The program address P is subtracted from the Branch destination K.
2. For values of K-P ranging from minus 7 to plus 7, a further test is made.
3. The value of K-P ranging from minus 7 to plus 7 is compared with the locator L and depth D.

It should be clear from this test that the Branch within the instruction stack can be forward or backward. Note also that the 6600 Branch instructions only allow for jumps to full word boundaries.

For jumps within the stack, no storage reference is initiated in order to preserve the current stack contents. Also, the initiation of new instructions will only occur if the bottom stack register supplies the instruction (Chapter VI). Therefore, program loops may be held in the stack in various forms.

Whether a "loop" within the stack or a jump out of the stack is performed, the new program address is set in the program address register P. This means that the instruction word defined by P can "float" up in the stack, only requiring the locator L for identification.

RETURN JUMP

The Return Jump instruction, 010jk, is unique in that two storage references are executed. The first stores an unconditional jump (0400) and the current address plus one (P + 1) in the upper half of address K. The second reference causes an instruction fetch from address K + 1. The effect of the Return Jump is shown in the diagram of Figure 70.

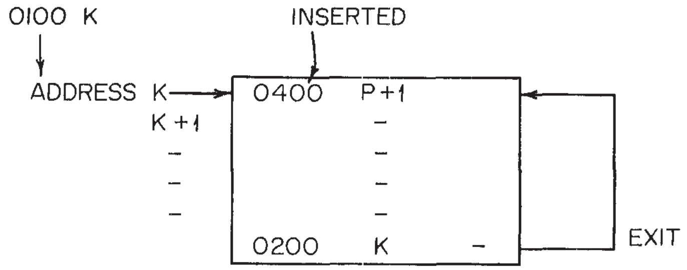


FIGURE 70

Return Jump "inserts" in exit unconditional Branch which is used by the subroutine to return to the originating routine. Since this "insertion" does not modify the lower half of the word at address K, this lower half can also be used by the originating routine to further identify the Branch source.

In any case, this instruction must be used with care since the return address left in address K can be destroyed by re-use of the subroutine after an unexpected interrupt. Although the Return Jump is a very useful and convenient instruction, it represents the type of program "self-modification" which requires careful handling.

Note: To such a pious statement as the preceding one, there is the temptation to claim foul. For example, what does "careful handling" mean? Or, alternatively, should it be used at all? The reader must simply determine the conditions which hold for the case in question. If a subroutine is to be used by many programs under interruption conditions, it is vulnerable.

The Return Jump instruction is not allowed to jump to the instruction stack because it must leave the return address in storage. The instruction stack does not return, of course, to central storage.

The steps involved in the Return Jump are as follows.

1. Read Return Jump.
2. Stop Instruction Issue.
3. Transfer P (contains P + 1) to S Register.
4. Transfer R (jump address K) to P.
5. Transfer P to Stunt Box register MO.
6. Transfer S to Storage Write Distributor and Force 0400 in Upper Bits of Distributor.
7. Increment P (Jump Address plus 1) and Transfer to Stunt Box Register MO.
8. Transfer MO and Stunt Box Tag for "Read Next Instruction" to Stunt Box Hopper.
9. Wait for Accept to Proceed.

J. ECS COUPLER-CONTROLLER

An optional addition to the 6600 computer system, the Extended Core Storage Unit, requires the equivalent of a CPU functional unit. This is called the ECS Coupler. Two CPU instructions are issued to the ECS Coupler, described below.

011jk	Read a block of length (Bj) + K words from ECS to Central Storage.
012jk	Store a block of length (Bj) + K words from Central Storage to ECS.

The starting address of Central Storage is defined by the contents of register A0. The absolute address in Central Storage is found by the sum of (A0) and RA. The starting address of Extended Core Storage is defined by the contents of register X0. The absolute address in ECS is found by the sum of (X0) and RAecs.

A length test is made at the beginning of the block transfer to determine if the field length for Central Storage FL or the field length for ECS FLecs will be exceeded by the transfer. If so, the instruction is aborted as described later.

Similar to the BRANCH Unit, all instruction "issues" are halted during an ECS execution. The reason for this is that the block transfer between Central Storage and ECS uses the Central Storage trunk system to full capacity in one direction of data flow.

Control is given to the ECS Coupler after the instruction is issued. Two restart possibilities exist. These are triggered by the End of Transfer signal or the Abort signal. Typically, the instruction is located in the upper, or left half, portion of the instruction word. On an End of Transfer signal, the remainder of the instruction word is ignored. On an Abort signal, the next instruction is taken from the lower, or right half, of the instruction word. This allows for separate treatment of the normal transfer and the aborted transfer.

Initial data entered into the ECS Coupler are the K field of the instruction, the contents of registers Bj, A0 and X0. The block transfer length is first determined by the sum (Bj) + K. Following, the test is made against FL and FLecs, shown below.

A0 + (Bj) + K-FL	tests for Central Storage.
X0 + (Bj) + K-FLecs	tests for ECS.

After the transfer length test is made, the ECS Coupler controls the block transfer, acting as the control interface between Central Storage and Extended Core Storage.

CONTROLLER

Extended Core Storage uses a "super-word," or sword, of 480 bits, with eight bits of parity. Therefore, addresses offered to the controller are required for every eighth Central Storage word. During a block transfer, the ECS Coupler delivers an initial address at the correct 60-bit word boundary, and subsequent addresses at sword boundaries.

The ECS Controller accepts addresses on a sword basis. By this means, each group of eight Central Storage words is transferred through the ECS Controller. On Read transfers the initial address is, in general, one of the eight 60-bit words within a sword, as shown in Figure 40 in Chapter IV.

The controller separates the sword address and word pointer for the READ operation. After the read reference is initiated and the sword is read into the Bank Register, the word pointer controls the beginning of the block transfer. Sixty-bit words are transferred, beginning at the word pointer location, and continue at minor cycle intervals to the end of the sword. After the first address, this is an eight-word cycle. In Figure 41 in Chapter IV is shown a series of eight-word cycles in a typical transfer.

This diagram illustrates the fixed time intervals associated with each sword address. The important thing is that each eight-word transfer is conducted at minor cycle intervals. It is possible to break in at sword boundaries to allow PPU references to Central Storage or to allow other access channels to reference ECS. Without such interruptions, a block transfer can proceed without any break, as shown. With any interruption, a penalty is paid in the control system and in the conflict of usage of ECS banks or Central Storage banks.

INTERRUPT

The ECS Coupler delivers sword addresses to the ECS Controller under the constraints mentioned above. Whenever a PPU reference to Central Storage is allowed to interrupt, a short restart time penalty is exacted. Another, more serious interruption can occur in the CPU. This is the appearance of an Exchange Jump. The effect of such a signal is to cause an abort of the entire transfer. It is assumed that the entire block transfer will be reinitiated in a later return to the program which was interrupted.

The ECS Coupler generates an address for every 60-bit Central Storage word in the block transfer. These addresses are entered in the Stunt Box mechanism with one assumption. This assumption is that the references will proceed in order with no conflict. This is insured because the beginning of operation is not allowed until all other Central Storage references are completed.

STORE

Store references are specially handled in the ECS Controller. A fixed time interval is required for words entering the ECS Controller to be stored. Once this fixed interval is over, no entries are honored. The final sword in a block transfer is handled satisfactorily in this manner since the words to be stored are delivered to the ECS Controller in consecutive minor cycles, until the block is completed. If the block is short of a sword boundary, the fixed time interval completes the operation correctly.

CENTRAL PROCESSOR CONTROL

VI

In the 6600 Computer, perhaps even more than in any previous computer, the control system is the difference. The essential premise of the Central Processor is "functional parallelism." Chapter V has dealt with the functional units. This Chapter will show the methods employed to control these units in parallel.

It is well to repeat the essentials of the "functional parallelism" of the 6600 Computer. These are:

- Separate functional units,
- Registers for operands, indexes, and addresses,
- Instruction Stack,
- Reservation control, or Scoreboard.

This last is the major part of the control system. There are, of course, other elements of the control system which are also essential. For example, the operating system for the 6600 provides overall control of job scheduling, allocation of CPU and storage, and the control of Input and Output.

A. EXCHANGE JUMP

The Central Processor is started, stopped, or otherwise interrupted by means of the Exchange Jump. This operation may be initiated by a Peripheral Processor or by the Central Processor. To initiate the operation, a

PPU executes the Exchange Jump referring to a Central Storage location as shown in Figure 71.

This location is the first of 16 words called the "exchange package." As seen, the contents of the 24 Central Processor Registers are copied into this package, together with other essential data including the program address, relative address and field length for Central Storage and ECS, error mode, and a pointer where applicable. The effect of executing the Exchange Jump instruction in the PPU is simply a pass. The effect, however, in the CPU is a series of the following steps.

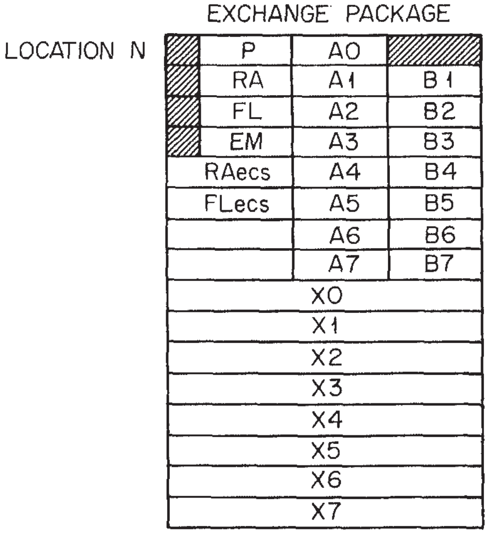


FIGURE 71

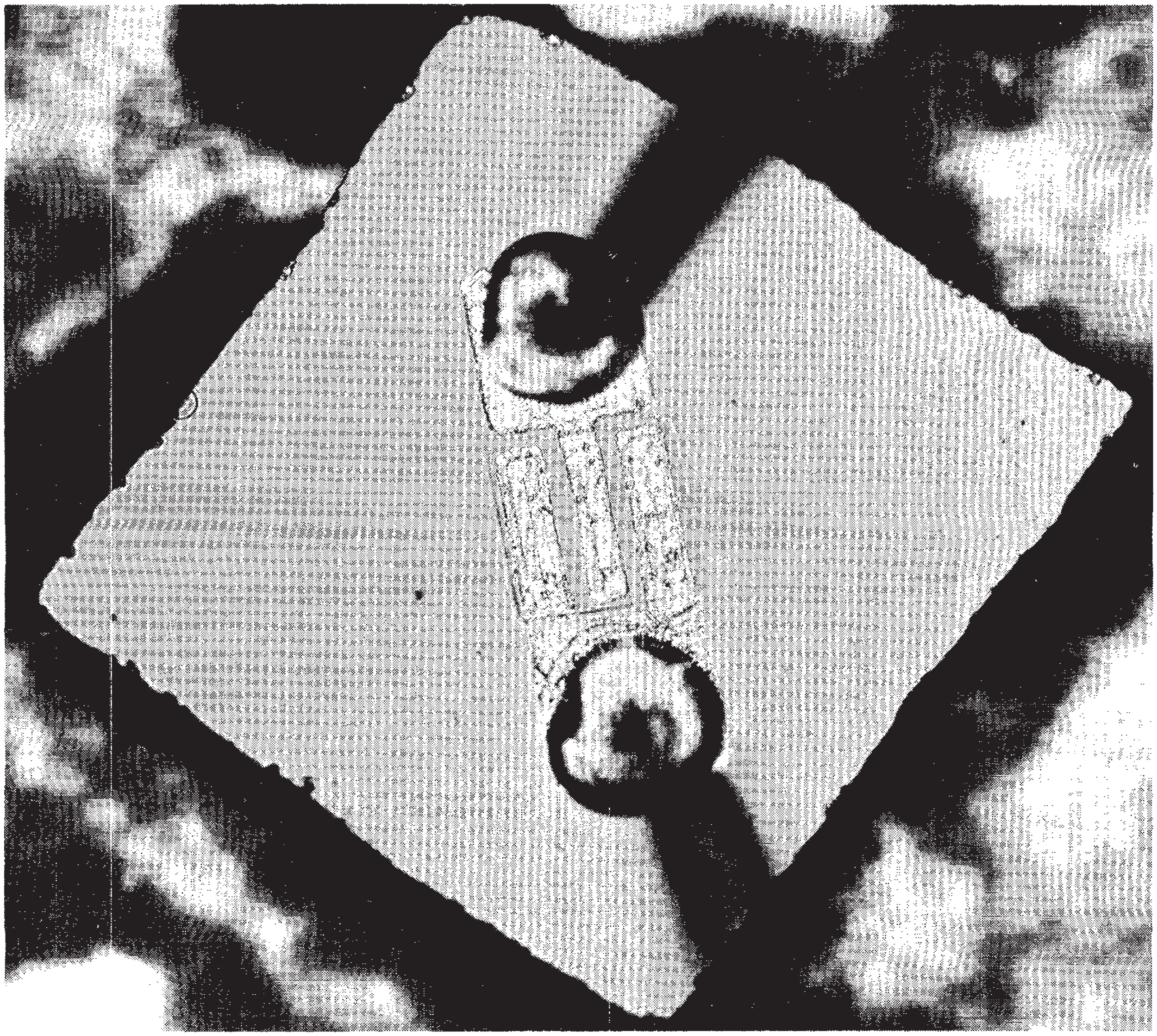
1. The CPU issues instructions up to, but not including, the next one located first in an instruction word.
2. All issued instructions are allowed to run to completion.
3. The CPU registers are then interchanged with the data stored in the exchange package.
4. The CPU is restarted at the location specified by the new contents of program address register, P.

These steps are completed in an uninterruptable sequence, taking a variable time for steps 1 and 2 and just over two microseconds for steps 3 and 4. Average interruption time is in the range of three to five microseconds.

The effect of step 1 above is sufficient to cover instruction combinations including Branches and also provides a clean starting point for restart at a later time. Step 2 insures that the integrity of the interrupted program is maintained. Step 3 is a special privilege provided by the Central Storage system, which allows readout of stored data and the write of new data in a single storage reference. Because there are only sixteen words to be interchanged, the interleaved bank structure of Central Storage is used efficiently. Finally, the new program is begun at the new location at (P) using the newly exchanged data. Note that the Exchange Jump produces new relative address and field length entries. In short, the entire "state" of the CPU is reset by the Exchange Jump.

As a hardware option the Central Processor can also initiate and then execute an Exchange Jump. Since this is a conceptually different situation from a PPU interruption, there are several changes from the simple Exchange Jump.

First, there is defined a "monitor" state in which the Central Processor may initiate jobs or tasks in a direct manner. The exchange package, in this case, is defined by the location (Bj) + K in instruction 013jk. The CPU may thus prepare a new task in a manner similar to the PPU, then initiate it. The CPU also inserts in the exchange package during the execution of the monitor program, a pointer to the "return" exchange package on completion or interruption of the task. Therefore, when the task signals completion by executing instruction 013jk, the designators j and k are ignored. Instead, the Exchange Jump is executed using the monitor address pointer to specify



Transistor "chip" showing surface metal pattern and connecting leads.

the location of the exchange package. The intended effect is to alternate between monitor and task. Note: *task* is used here to define programs which require operating system assistance to initiate. A task may be a portion of a user program, a "public" library program, or the operating system itself. A task may be interrupted in some "time slicing" manner.

The PPU is also able, within the same option, to control Exchange Jumps in the manner described above. For this purpose a conditional Exchange Jump may be executed. The effect is to cause a "return" to the monitor if the CPU is not already operating in the monitor state. If the CPU is operating in the monitor state, no action is taken. The operating system software in this case provides for a confirmation by the initiating PPU, using a message area in Central Storage. The PPU, by program, repeats the conditional Exchange Jump until successful.

The optional Exchange Jumps described above are especially useful and interesting in conjunction with Extended Core Storage and the compatible 6500 Computer which contains two smaller CPU's. In the latter case, each CPU can interrupt the other and itself in the manner described. The advantages of two CPU's, even though slower than a 6600 CPU, are particularly sensitive to the amount of storage and to the ability of each CPU to initiate new tasks. The optional Exchange Jumps are most useful in this case.

The Exchange Jump operation is a key mechanism of the 6600 operating system. With it, a PPU may act as system monitor, scheduling and interrupting the CPU. Similarly, with the hardware option the CPU can take over the Central portion of the system monitor while still deferring Input-Output and external interrupt handling to the PPU's.

B. INSTRUCTION FETCH

Following any Exchange Jump, the new contents of the program address register P are used to locate the first instruction word. This address is sent to the Central Storage Stunt Box modified by the new relative address, (RA) + (P). The program address is also tested against the Central Storage Field Length FL. If the program address exceeds the field length, all zeroes are read from Central Storage. If the error mode is set to abort on such a fault, the program branches to Relative Address RA and halts indicating the error. If the error mode is not set, the new instruction of all zeroes is "executed," producing a halt. Note that the error mode conditions cover the detection of infinity and indefinite in floating point operands as well as the storage reference out of bounds described here.

Assuming a normal program start, the first instruction word enters a buffer register located at the bottom of the instruction stack, briefly discussed in Chapter V. This is shown in Figure 72.

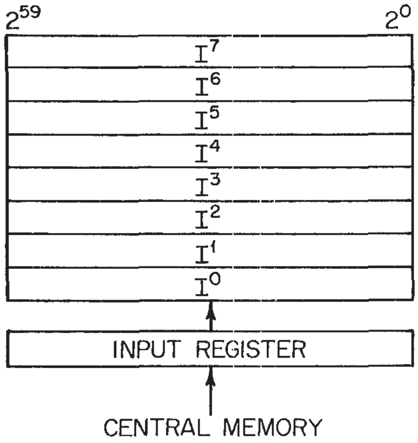


FIGURE 72 Instruction stack.

Immediately on entering the bottom register, the first, or left-most, instruction is transferred to a series of instruction registers, U0 through U2. As this transfer occurs, another instruction fetch is initiated. The condition for this step is simply that the left-most instruction is being transferred for execution. In later discussion it should be clear how this satisfies all instruction fetch conditions to Central Storage except the start after Exchange Jump.

Instruction words are made up of four "parcels" of fifteen bits each. The first instruction in a word uses parcel 0 for short format or parcel 0 and 1 for long format, as shown in Figure 73.

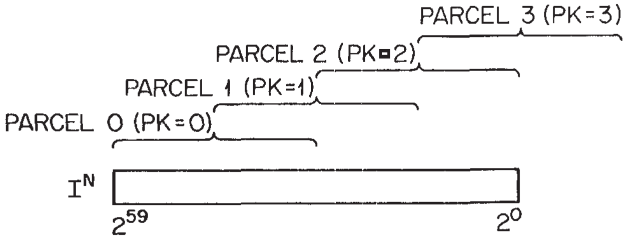


FIGURE 73 Instruction word parcels.

In order to cover the long format, thirty bits are extracted from the instruction word on every minor cycle. For short formats the second half is ignored. For long formats an extra minor cycle is spent skipping the second half.

Instructions are loaded in register U1 by two paths as shown in Figure 74 (page 122).

From odd levels of the instruction stack instructions enter U1 directly. From even levels of the instruction stack instructions enter U1 via register U0. This form of logic is a result of the instruction stack data movement. Instructions are entered at the bottom of the stack following a shift maneuver in the stack. Whenever a new instruction fetch is initiated, the contents of the stack are "inched" upward, one register every half minor cycle. As a

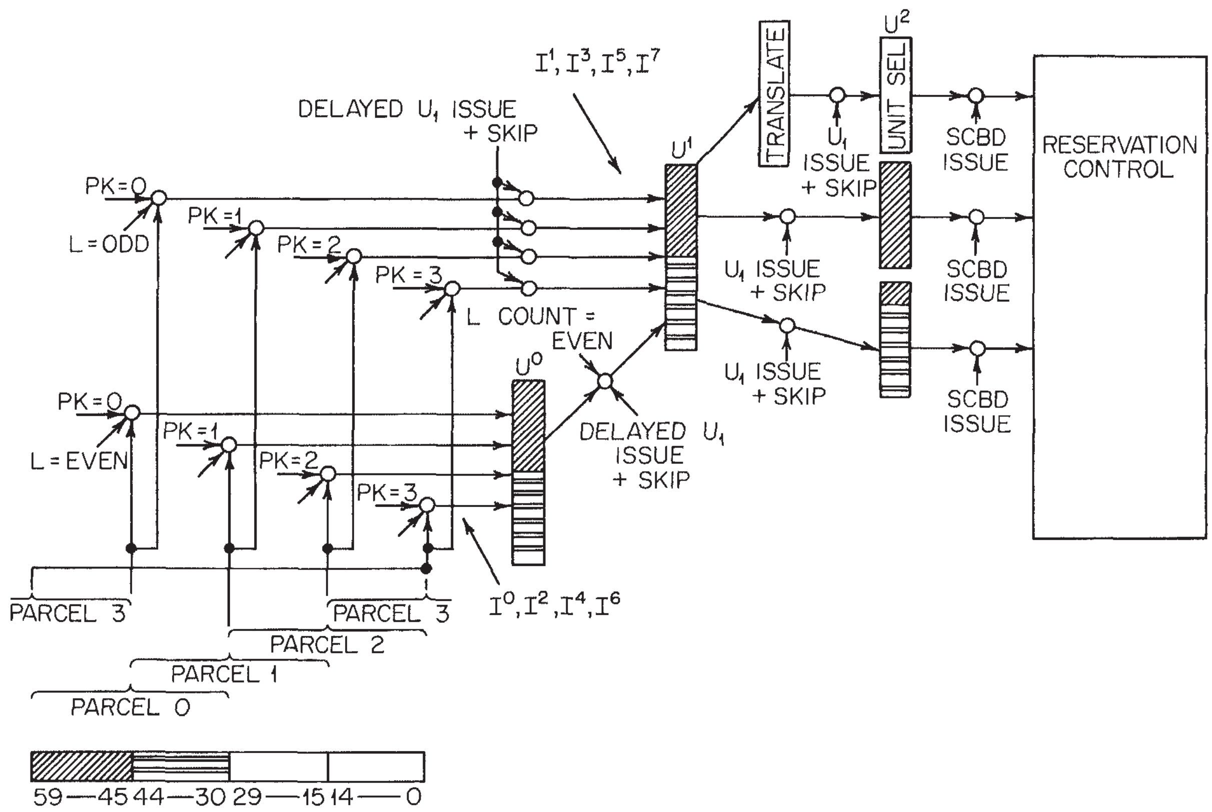

I^1, I^3, I^5, I^7
DELAYED U1 ISSUE + SKIP
PK=0
L=ODD
PK=1
PK=2
PK=3
L COUNT = EVEN
U^1
TRANSLATE
U1 ISSUE + SKIP
U^2
UNIT SEL
SCBD ISSUE
RESERVATION CONTROL
U1 ISSUE + SKIP
SCBD ISSUE
U1 ISSUE + SKIP
SCBD ISSUE
U^0
DELAYED U1 ISSUE + SKIP
PK=0
L=EVEN
PK=1
PK=2
PK=3
I^0, I^2, I^4, I^6
PARCEL 3
PARCEL 3
PARCEL 2
PARCEL 1
PARCEL 0
59—45 44—30 29—15 14—0

FIGURE 74 Parcel counter.

natural result of this design, each odd level is available on a specified half cycle, and each even level on the opposite half cycle.

Once the instruction is brought to register U1, the function translation can begin. This translation is completed during the transfer from register U1 to register U2. In this location, the command ISSUE is the primary control over instruction execution. In register U2, the instruction details are examined for conflict in the use of registers or functional units. If no conflict exists, the instruction is ISSUEd to the appropriate functional unit for execution. Another instruction can then be brought to register U2 for ISSUE in one minor cycle. If, however, a conflict exists which would delay execution, the instruction may not be ISSUEd. This is discussed in detail in a later section.

Instructions continue to be fetched sequentially from Central Storage until a Branch causes control to transfer within or outside the instruction stack. If the branch is to one of the "old" instructions currently in the instruction stack, the sequence can be best described with the following example.

Location LOOP	Instruction Word A
Location LOOP + 1	Instruction Word B
Location LOOP + 2	Instruction Word C Contains—GO TO LOOP IF . . .

This example shows a portion of a program contained in three instruction words A, B, and C located in relative location LOOP, LOOP + 1 and

LOOP + 2. Instruction word C contains a branch instruction, GO TO LOOP IF . . . , which causes control to transfer back to instruction word A if the condition is satisfied. Instruction fetch of each of these instructions is initially accomplished from Central Storage. In fact, because the first pass through this program "loop" requires that the first instruction contained in instruction word C be transferred to U2, an additional fetch is initiated. The result is that an additional instruction word D is entered in the lowest level of the instruction stack before the Branch is taken. After word D is entered into the stack, the "loop" can be held without further entries. This could be called a form of "look behind."

Instruction fetch and execution from the instruction stack is faster than from Central Storage for three reasons.

- Access time to the instruction stack is short.
- Test of Central Storage busy adds time, as does the summation of relative address RA with the Central Storage address.
- Use of the instruction stack removes a Central Storage reference.

Following any Branch, whether to Central Storage or to the instruction stack, a time penalty must be paid in loading the U0, U1, and U2 registers. This penalty is counted in the time given for Branch instructions. Subsequent instructions, of course, can be brought through these registers with no apparent time penalty, as if in an assembly line, or pipe line, flow. The test for looping back into the instruction stack is performed by the Branch unit while its "partner" unit is testing for the specified condition. Two quantities are maintained to perform this loop test, a location counter L and a depth counter D. The test first determines if the destination is within seven words of the current program address, P. The location of the current program address is also tracked by location counter L in case it isn't at the bottom of the instruction stack. After the determination "within seven," a test is made if the destination is validly within the instruction stack, a test involving both L and D.

The logical mechanism for detecting that the first instruction of the bottom stack word was being transferred did not bend, in the design, to prediction. Therefore, the condition for instruction fetch from Central Storage is accomplished in one minor cycle, and two additional cycles are required to enter the Central Storage Stunt Box. Central Storage requires five minor cycles for access, for a total of eight minor cycles for repeated instruction fetches from Central Storage.

C. INSTRUCTION ISSUE

Part of the time required for any Branch instruction is taken up in transferring the destination instruction through the series of instruction registers U0 through U2. This time is not wasted, however, because the new instruction is translated and analyzed in this trip. Following instructions are also brought through the same path in minor cycle intervals.

From instruction register U2, the instruction is *issued* to the functional unit designated. Only two restrictions are made on this issue.

- The Register designated for the result must not be reserved for a result by a previous instruction.
- The functional unit designated must not be busy.

Because these two conditions are fairly simple to establish, the instruction issue decision is made quickly. Instructions may be issued at minor cycle intervals dependent only on these two conditions.

The "supply" of instructions in the "pipe" leading to register U2 is synchronized with each issue signal. Each time an ISSUE command is given, a 30-bit quantity is transferred to the designated functional unit. Most units, of course, only utilize 15 bits. For 30-bit formats, however, the second parcel is also transferred at the ISSUE command. Following this minor cycle, a SKIP cycle is required in order to move beyond the second parcel of the issued instruction.

The SKIP command is also useful simply for bringing instructions from the instruction stack to register U2 after Branch. This is true since the only action required of the SKIP command is to cause all the housekeeping tasks of moving new instructions into position without actually issuing any. A typical sequence of minor cycles is given below to show the usage of these commands.

```
ISSUE BRANCH
   Wait Branch
SKIP                              Loads 1st instruction in U1
SKIP                              Loads 2nd half of 1st instruction
                                     in U1 and 1st instruction in U2

ISSUE 1st instruction
SKIP                              Assumes 1st instruction is 30-
                                     bit format

ISSUE 2nd instruction
ISSUE 3rd instruction
   Wait Next Instruction Word
```

This example shows an instruction word with a 30-bit instruction followed by two 15-bit instructions. It shows the usage of the SKIP commands which merely control the instruction "pipe" during filling and skipping steps.

Handling of the test for Branching in the instruction stack is complicated by the control of instruction issue. This complication arises from the conditional Branch instructions which can do one of the following:

Loop	—A conditional Branch, condition met, in the stack.
Jump	—An unconditional Branch, or condition Branch, condition met, not in the stack.
No Branch	—A conditional Branch, condition not met.

To test for the destination in the Stack, the contents of the program address register, P, must remain set equal to the address which contains the Branch instruction. Therefore, P is not changed until after the "third parcel" instruction is issued, that being the last possible location of a 30-bit Branch instruction in a word.

An additional problem of the Branch is the condition of the issue control mechanism following a "fall-through," or no-branch, condition. Instructions, following the Branch instruction in the same instruction word, are held in the registers U1 and U2 after the issue of the Branch instruction. Then, if the branch condition is not met, these instructions are brought to ISSUE in the normal manner. As a result, the control over the instruction stack output must track directly with the ISSUE command.

D. SCOREBOARD

A unique and essential part of the 6600 Central Processor control is the Unit and Register Reservation Control, or the Scoreboard. What is intended by this design is the simultaneous operation of functional units on a single instruction stream. Many operations in these units are quite independent of others, due to the relative simplicity of the instructions. It is often particularly apparent that a sequence of arithmetic or logical operations can be executed simultaneously with a sequence of control or housekeeping operations. Again, examples will be shown in which considerable overlap is possible even in the single sequence.

One major premise of the Scoreboard design is that each new instruction be issued to its functional unit as early as possible in order to allow following instructions to be issued. In some cases, an issued instruction may be held up after issue awaiting input operands, while a following instruction may proceed without restraint.

Three types of conflict can be described in the usage of functional units and registers, which must be resolved by the Scoreboard.

1. First Order Conflict
   This is a conflict between instructions which require the same functional unit or the same result registers.

   Example one. Functional unit conflict

$$X6 = X1 + X2$$
$$X5 = X3 + X4$$

Both instructions use the Add functional unit, a situation in which the second instruction must wait for the first to be completed.

In the case of multiply or increment instructions, two units are provided reducing the probability of this conflict.

Example two. Result register conflict

X6 = X1 + X2
X6 = X4 * X5

Both instructions call for register X6 for the result, another situation in which the second instruction must wait for the first to be completed. Although the example shown is a trivial case, it will be seen in later discussion that many nontrivial cases are possible.

The control over this conflict is simply that of not issuing the second instruction until the first is completed. At issue time, the condition must be determined early enough to stop the ISSUE command.

2. Second Order Conflict
   This conflict occurs when an instruction requires the result of a previously issued, and as yet uncompleted, instruction as a source or input operand.

   Example:

   X6 = X1 + X2
   X7 = X5/X6

   Register X6 in this example is used as the result of the Add instruction and then as the divisor in the Divide instruction. The second instruction is issued but held in the Divide Unit until result X6 is ready.

   The second order conflict does not halt issuing of instructions but is resolved by the scoreboard control over the functional unit.

3. Third Order Conflict
   This conflict occurs when an instruction is called on to store its result in a register which is to be used as an input operand for a previously issued, but as yet unstarted, instruction.

   Example:

   X3 = X1/X2
   X5 = X4 * X3
   X4 = X0 + X6

   In this example the third order conflict on the use of register X4 is a direct result of a second order conflict on register X3. Because the instructions are issued on consecutive minor cycles and because the

Add function is much faster than Divide or Multiply, the addition is accomplished and ready for entry in the result register X4 well in advance of the start of Multiply. The second order conflict on register X3 causes the Multiply to hold until that input operand is ready. This holds up the entry of register X4 into the Multiply Unit also.

Third order conflicts are resolved by holding the result in the functional unit.

Scoreboard control thus directs the functional unit in starting, obtaining its operands, and storing its result. Each unit, once started, proceeds independently until just before the result is produced. The unit then sends a signal to the Scoreboard requesting permission to release its result to the result register. The Scoreboard determines that the path to the result register is clear and signals the requesting unit to release its result. The releasing units reservations are then cleared, and all units waiting for the result are signaled to read the result for their respective computations.

DESIGNATORS

The Scoreboard gets its name from the number of designators and identifiers used in performing the job of reservation control. Figure 75 on page 129 diagrams the number of designators associated with one functional unit. Shown is the Add Unit which is given the number 17 with function designators, reservation identifiers and flags as described below.

Fm —Function to be performed (ADD)
Fi —Designates register Xi for result
Fj —Designates register Xj as addend
Fk —Designates register Xk as augend

Qj —Identifies the functional unit, by number, producing a result to be used as addend
Qk —Identifies the functional unit, by number, producing a result to be used as augend

Read Flag j —A single-bit flag indicating that the addend is ready
Read Flag k—A single-bit flag indicating that the augend is ready

Xi —Identifies that the Add Unit, number 17, has reserved register Xi for its result. (Bi and Ai for other units)

All functional units are assigned a number to be used in the identifiers Q for the units, X for the operand registers, B for the increment registers, and A for the Address registers. These numbers are assigned as follows:

Designator (Octal)	Functional Unit
00	Branch
01	Increment 1
02	Increment 2
03	Shift
04	Boolean
05	Divide
06	Multiply 1
07	Multiply 2
10	—
11	Read Storage Channel 1
12	Read Storage Channel 2
13	Read Storage Channel 3
14	Read Storage Channel 4
15	Read Storage Channel 5
16	Fixed Add
17	Add

The Scoreboard operation is described in two parts; first, placing reservations, and second, directing the read operand and store result operations of each unit.

PLACING RESERVATIONS

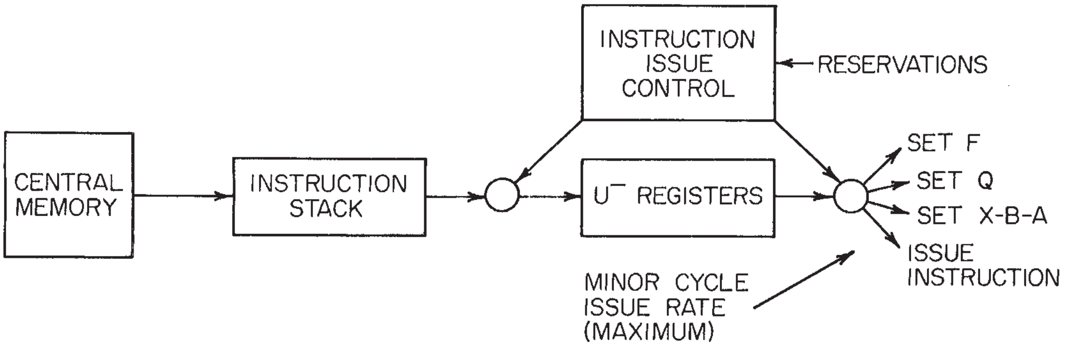
This portion of the Scoreboard operation is executed in four sequential steps at the time an instruction is issued. These steps are as follows.

1. Reserve the functional unit, Set its "busy" flag, and enter the operating mode (fm).
2. Set the register designators in the functional unit, Fi, Fj and Fk.
3. Enter any previous result reservations on the entry operands, Qj and Qk.
4. Set the result register identifier, Xi, Bi, or Ai with the functional unit number.

Step one, SET UNIT BUSY, is rather straightforward except as the determination of unit "busy" is made. As an example, two consecutive instructions to the same unit must be handled such that the second instruction ISSUE is disallowed. Since these are one minor cycle apart, the setting of unit "busy" flag by the first instruction followed by the test for busy by the second instruction must be accomplished in one minor cycle.

Step two, SET F, transfers the i, j, and k fields of the instruction to the designators of the functional unit. These are then used to designate operand and result registers to be used by the unit. Figure 76 shows how these designators are transferred from U1 to U2 and then to the respective functional units.

Notice that the Branch instructions cause a right shift of the designators i and j in register U1 to j and k respectively in register U2. This ma-


INSTRUCTION ISSUE CONTROL
RESERVATIONS
CENTRAL MEMORY
INSTRUCTION STACK
U⁻ REGISTERS
SET F
SET Q
SET X-B-A
ISSUE INSTRUCTION
MINOR CYCLE ISSUE RATE (MAXIMUM)

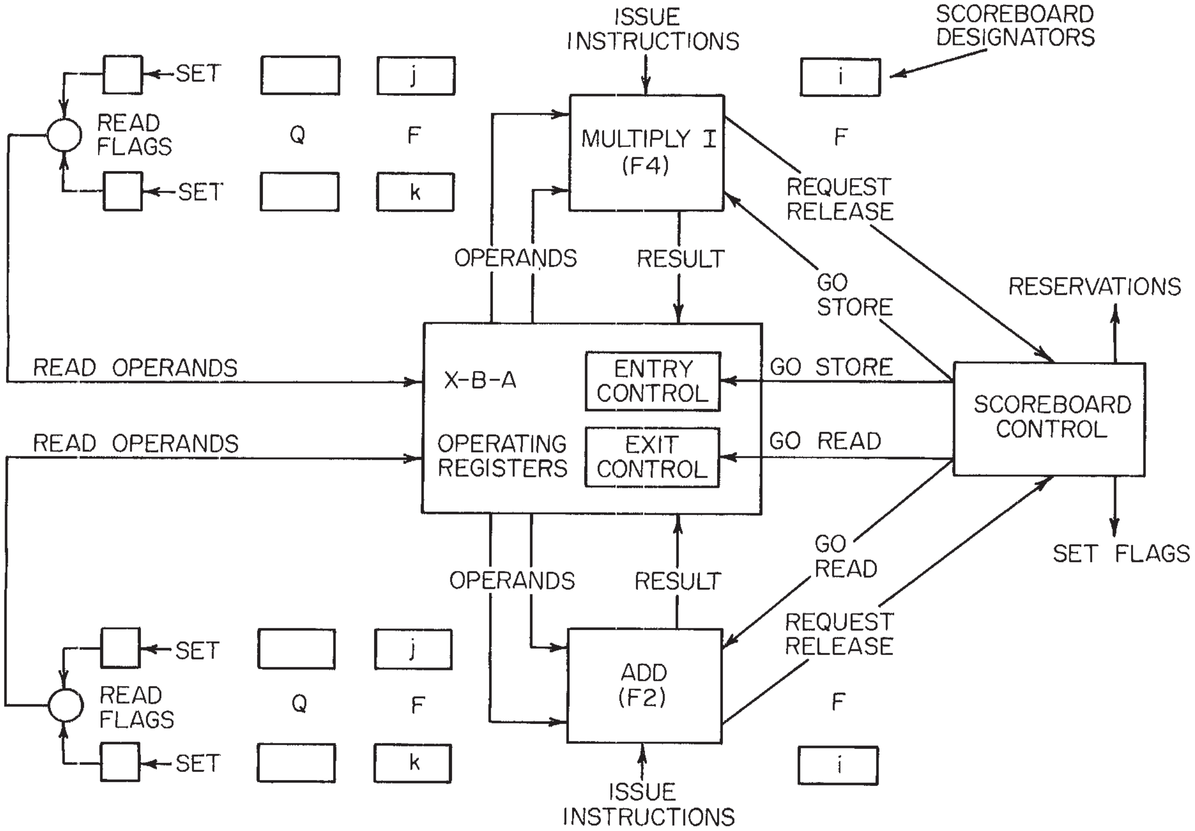

ISSUE INSTRUCTIONS
SCOREBOARD DESIGNATORS
SET
READ FLAGS
SET
Q
F
j
k
i
F
MULTIPLY I (F4)
REQUEST RELEASE
GO STORE
OPERANDS
RESULT
RESERVATIONS
READ OPERANDS
X-B-A
ENTRY CONTROL
GO STORE
SCOREBOARD CONTROL
READ OPERANDS
OPERATING REGISTERS
EXIT CONTROL
GO READ
GO READ
SET FLAGS
OPERANDS
RESULT
REQUEST RELEASE
SET
READ FLAGS
SET
Q
F
j
k
ADD (F2)
F
i
ISSUE INSTRUCTIONS

FIGURE 75 Reservation designators.

neuver is convenient to allow a direct usage of the Increment and Fixed Add Units as partner units to the Branch unit for conditional branch instructions.

Step three in placing reservations, SET Q, is essentially a copying operation from one of the 24 XBA identifiers related to the 24 operating registers. The identifier contains the functional unit number of the unit which has reserved that register for a result. Since there are usually two Q identifiers, one for each input operand, there may be two independent settings. See Figure 77. Following this step, the essential link between a previous result and an input operand is established.

Step four, the final step in placing reservations, SET XBA, places the

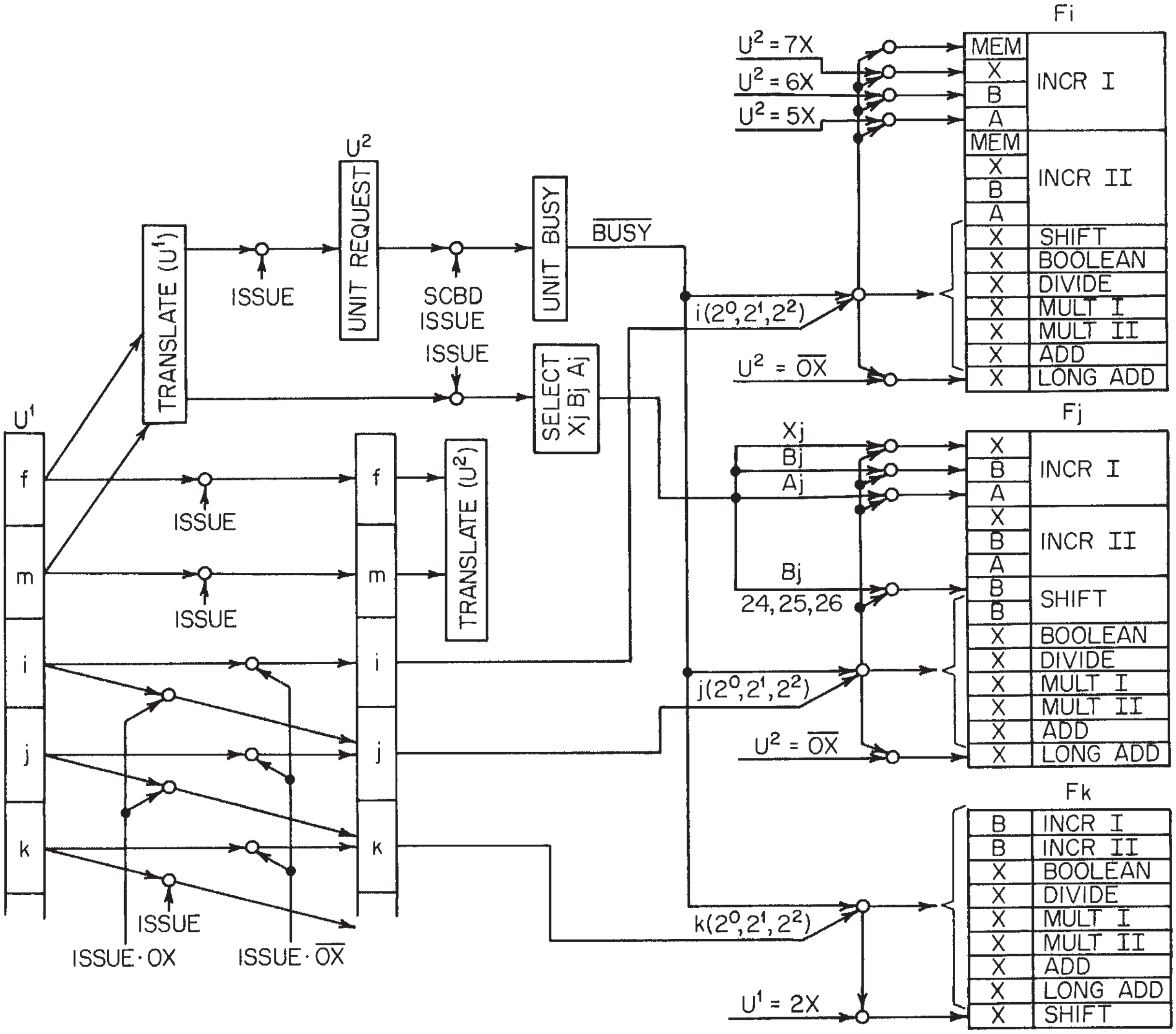

Fi
U² = 7X
U² = 6X
U² = 5X
MEM
X
B
A
INCR I
MEM
X
B
A
INCR II
X SHIFT
X BOOLEAN
X DIVIDE
X MULT I
X MULT II
X ADD
X LONG ADD
U²
UNIT REQUEST
UNIT BUSY
BUSY
TRANSLATE (U¹)
ISSUE
SCBD ISSUE
ISSUE
SELECT Xj Bj Aj
i(2⁰,2¹,2²)
U² = OX
U¹
f
m
i
j
k
ISSUE
ISSUE
TRANSLATE (U²)
Fj
Xj
Bj
Aj
X
B
A
INCR I
X
B
A
INCR II
Bj
24,25,26
B
B
SHIFT
X BOOLEAN
X DIVIDE
X MULT I
X MULT II
X ADD
X LONG ADD
j(2⁰,2¹,2²)
U² = OX
Fk
B INCR I
B INCR II
X BOOLEAN
X DIVIDE
X MULT I
X MULT II
X ADD
X LONG ADD
X SHIFT
k(2⁰,2¹,2²)
U¹ = 2X
ISSUE
ISSUE·OX
ISSUE·OX

FIGURE 76 Set F

functional unit number in the identifier associated with the result register. Translations of the function to be performed are necessary in order to select the correct register group, X, B or A, along with the correct register in the group. Note that the unit numbers were chosen such that only two bits are necessary for the B and A registers, whereas four bits are needed for the X registers. Only three units cause results in B and A registers, whereas up to ten "units" cause results in X registers. This, of course, includes the Read Storage channels into registers X1 through X5. The unit number generator produces the necessary unit numbers to be entered. In the case of Read Storage instructions, which produce a new result in registers A1 through A5, the A identifier is set with the Increment Unit number, and the partner X identifier is set to the Read Storage Channel number. See Figure 78.

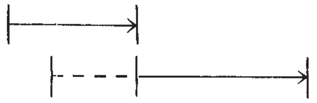
SET READ FLAGS

After issuing the instruction and placing reservations, the Scoreboard proceeds to control the functional unit in reading operands and storing results. The first activity in the functional unit is the simultaneous "reading" of input operands. The unit may not start until both operands are ready to be read. Both Read Flags must, therefore, be set.

The conditions for setting a Read Flag are determined by the Q identifier associated with that input operand and by the Release signal from the functional unit identified by Q. The effect is to link the result of the previous operation with the input operand. The example used in the description of second order conflict is repeated here to show the effect.

X6 = X1 + X2

X7 = X5/X6



When the second instruction is issued, the third step in placing reservations, SET Q, causes the unit number found in identifier X6 to be placed in the Qk identifier for the divisor. The unit number is, of course, 17 for the Add Unit, having been placed there at the time of issue of the first instruction.

When the Add Unit requests release and receives permission, a Release

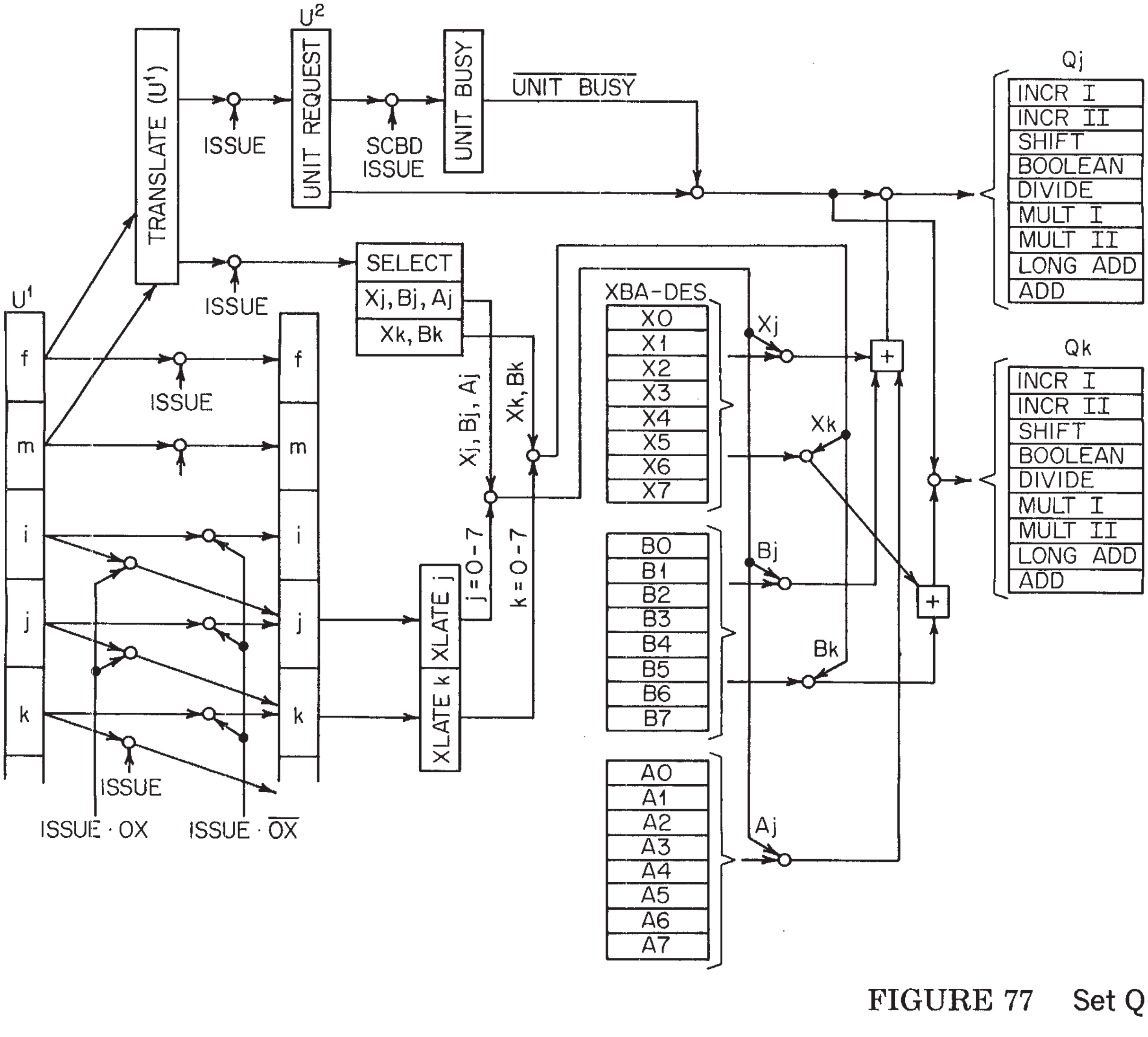

U²
TRANSLATE (U¹)
ISSUE
UNIT REQUEST
SCBD ISSUE
UNIT BUSY
UNIT BUSY
SELECT
Xj, Bj, Aj
Xk, Bk
ISSUE
U¹
f
m
i
j
k
ISSUE
f
m
i
j
k
ISSUE
ISSUE · OX
ISSUE · OX
Xj, Bj, Aj
Xk, Bk
j = 0 - 7
k = 0 - 7
XLATE j
XLATE k
XBA-DES
X0
X1
X2
X3
X4
X5
X6
X7
B0
B1
B2
B3
B4
B5
B6
B7
A0
A1
A2
A3
A4
A5
A6
A7
Xj
Xk
Bj
Bk
Aj
+
+
Qj
INCR I
INCR II
SHIFT
BOOLEAN
DIVIDE
MULT I
MULT II
LONG ADD
ADD
Qk
INCR I
INCR II
SHIFT
BOOLEAN
DIVIDE
MULT I
MULT II
LONG ADD
ADD

FIGURE 77 Set Q

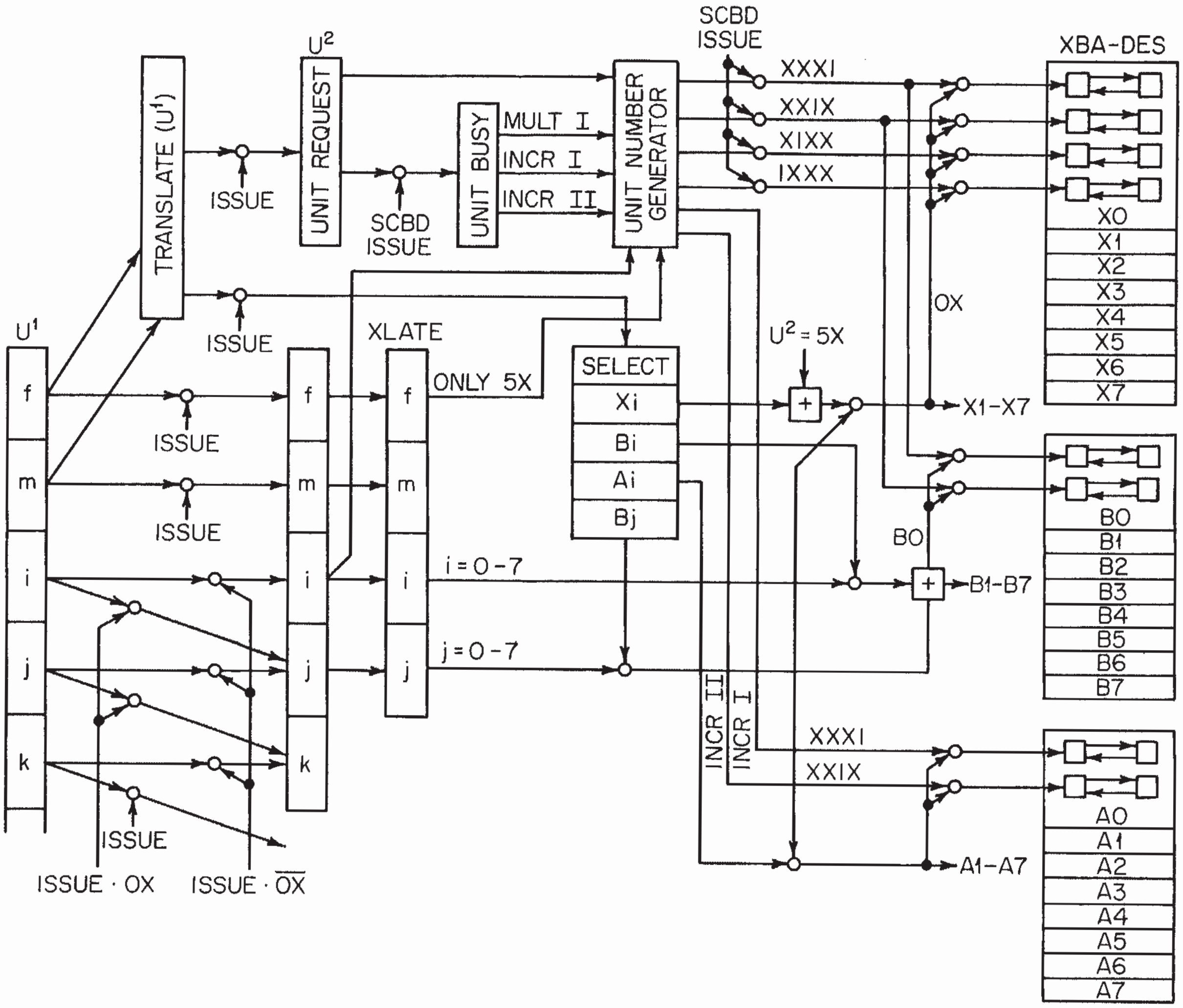

U²
UNIT REQUEST
TRANSLATE (U¹)
ISSUE
SCBD ISSUE
UNIT BUSY
MULT I
INCR I
INCR II
UNIT NUMBER GENERATOR
SCBD ISSUE
XXXI
XXIX
XIXX
IXXX
XBA-DES
X0
X1
X2
X3
X4
X5
X6
X7
0X
U¹
f
m
i
j
k
XLATE
ONLY 5X
SELECT
Xi
Bi
Ai
Bj
U² = 5X
X1-X7
B0
B0
B1
B2
B3
B4
B5
B6
B7
B1-B7
i = 0-7
j = 0-7
INCR II
INCR I
XXXI
XXIX
A0
A1
A2
A3
A4
A5
A6
A7
A1-A7
ISSUE · 0X
ISSUE · $\overline{0X}$

FIGURE 78 Set XBA

signal is sent to all units, among them the Divide Unit. This is shown in Figure 79.

All Release signal lines are shown as they appear to the divisor input to the Divide Unit. The case in point is the Release signal from the Add Unit, which AND's with the translation of the unit number held in Qk for the Divide Unit. Since the Qk identifier can hold only one unit number, only one Release signal is selected. Assuming the Q identifier is set to zero, meaning no wait is necessary, the Read Flag is set immediately after issue.

Figure 79 also shows how the Release signals are actually sent to all Read Flag networks. The example of Release for the Read Storage Channel 5 is shown going to the Q translation for unit number 15 on all nine units. This example appears to skip some Read Flag circuits, but it should be remembered that the X Registers are not connected as input operands in every combination to all units. The k operand in the Increment Units and the j operand in the Shift Unit are noteworthy.

When both Read Flags are set on any unit, the unit may be expected to start. However, it should be clear that several units could reach this condition simultaneously. For units which share data trunks (Chapter V),

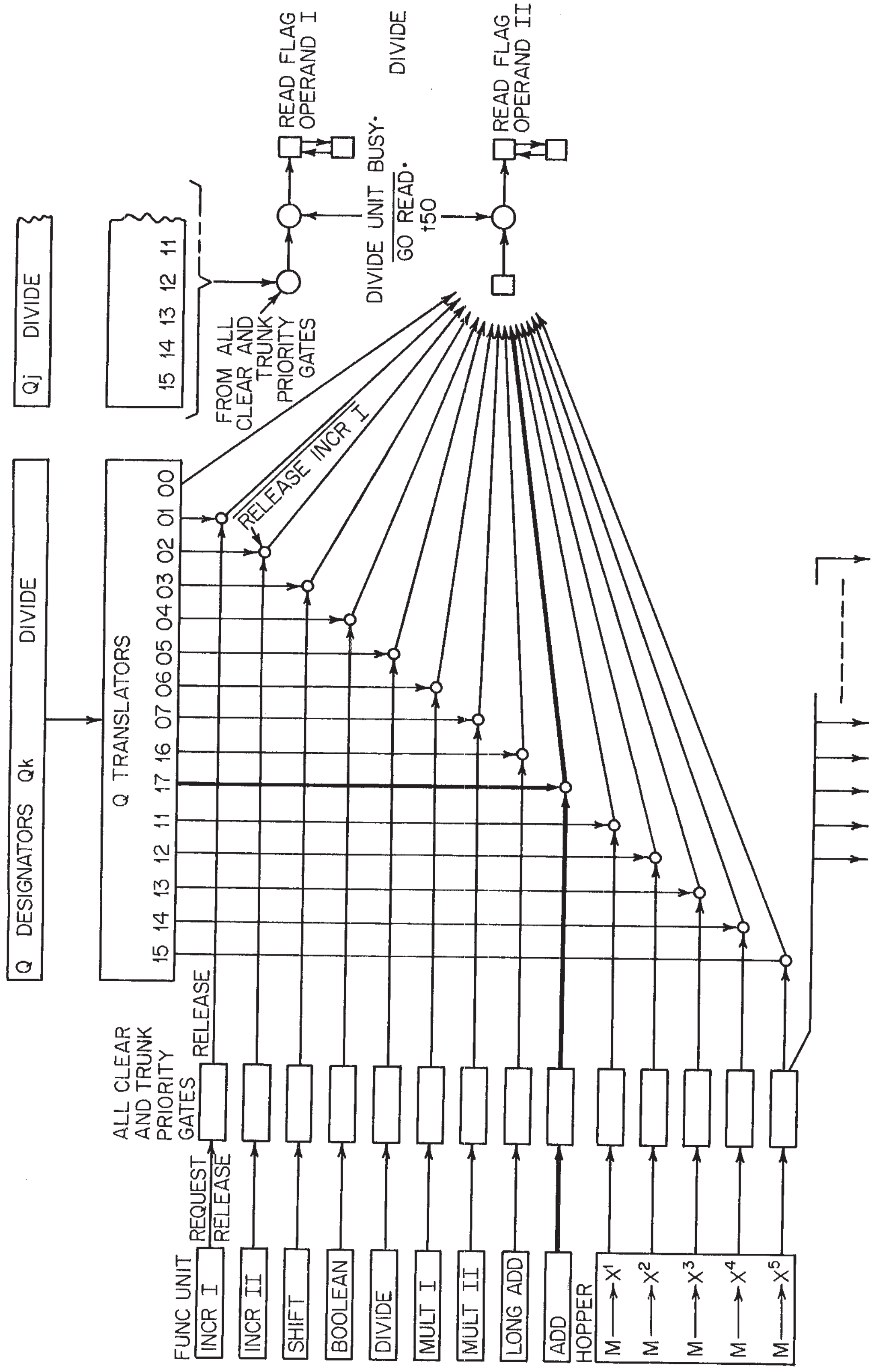

Q DESIGNATORS Qk DIVIDE
Qj DIVIDE
Q TRANSLATORS
15 14 13 12 11 17 16 07 06 05 04 03 02 01 00
15 14 13 12 11
FROM ALL CLEAR AND TRUNK PRIORITY GATES
RELEASE INCR I
READ FLAG OPERAND I
DIVIDE
DIVIDE UNIT BUSY·
GO READ·
t50
READ FLAG OPERAND II
TO OTHER UNITS
ALL CLEAR AND TRUNK PRIORITY GATES
RELEASE
REQUEST
RELEASE
FUNC UNIT
INCR I
INCR II
SHIFT
BOOLEAN
DIVIDE
MULT I
MULT II
LONG ADD
ADD
HOPPER
M → X1
M → X2
M → X3
M → X4
M → X5

FIGURE 79 Set read flags divide unit.

this would mean simultaneous traffic on the trunk. Therefore, the data trunk priority condition also controls the start of the unit.

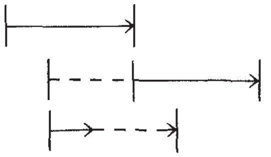
RELEASE

An additional factor in the Scoreboard control has to do with the Release signal. The release of the result to the result register would be uncomplicated were it not for the third order conflict and the result data trunk conflict. The third order conflict described before is repeated here.

X3 = X1/X2

X5 = X4 * X3

X4 = X0 + X6



In this example the third instruction is completed well in advance of the first two but cannot release its result to register X4 until the previous Read is accomplished.

Close examination of the example will show that the Read Flag for Multiply j input, corresponding to the X4 input, is set and simply waiting for the k Read Flag. The k Read Flag is held up by a second order conflict. Note that the third instruction would not be issued if the Multiply j input Read Flag were not set since that would indicate a previous result to register X4 not yet completed.

This is a form of proof that a Read Flag can be cause to hold up the Release signal. Each register can be described as "all clear" if no Read Flags are set corresponding to that register. To generate the All Clear for each register, the Fj and Fk designators are translated to the register number and ANDed with the associated j or k Read Flag.

These ALL CLEAR signals for each register are then combined with the translation of the result designator, Fi, for the unit to determine whether the unit should be allowed to release its result.

Assuming that the unit is held back, some time later the Read Flag will be cleared as a result of its unit starting, thereby clearing the flag. The entire case is presented in Figure 80.

E. REGISTER ENTRY/EXIT CONTROL

The secondary control over data entering and leaving the registers is provided by a rather simple system. Entry to the X, B or A registers is a direct result of the Release mechanism described in the section on the Scoreboard. A "GO STORE" signal is generated by the release mechanism, directing the requesting functional unit to transmit its result to the registers via its result trunk. At the same time, the result designator, usually Fi, is also sent to the register end of that trunk. This designator is translated and control is initiated to clear the result register and transfer the data from the

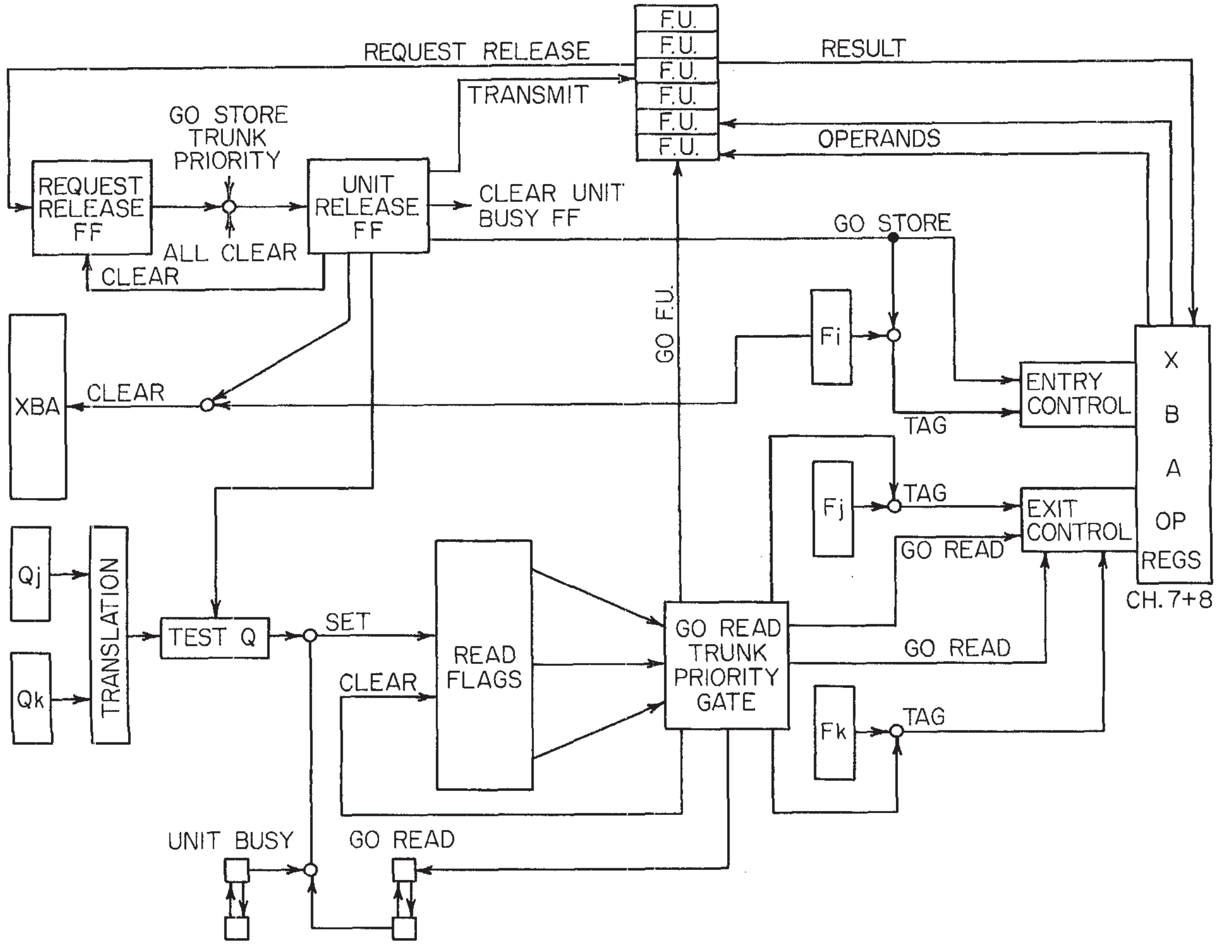

F.U.
F.U.
F.U.
F.U.
F.U.
F.U.
REQUEST RELEASE
TRANSMIT
RESULT
OPERANDS
GO STORE TRUNK PRIORITY
REQUEST RELEASE FF
UNIT RELEASE FF
CLEAR UNIT BUSY FF
ALL CLEAR
CLEAR
GO STORE
GO F.U.
Fi
Fj
Fk
TAG
TAG
TAG
ENTRY CONTROL
EXIT CONTROL
X B A OP REGS
CH. 7+8
XBA
CLEAR
GO READ
GO READ
Qj
Qk
TRANSLATION
TEST Q
SET
CLEAR
READ FLAGS
GO READ TRUNK PRIORITY GATE
UNIT BUSY
GO READ

FIGURE 80 Request release block diagram.

trunk to the correct result register. This entire operation is dependent on the fixed-time nature of the synchronous design of control functional units and data trunks. Figure 81 shows some detail of the entry control for registers X, B, and A. Note that the central storage entry trunk is also used by exchange jump for initial loading of these registers. Also note the five D registers used to hold the read operands from central storage in case of the third order conflict case described previously.

In a fixed, synchronous manner similar to the entry control, the register exit control provides for the transfer of data to the data trunks. Seven trunks are controlled as shown.

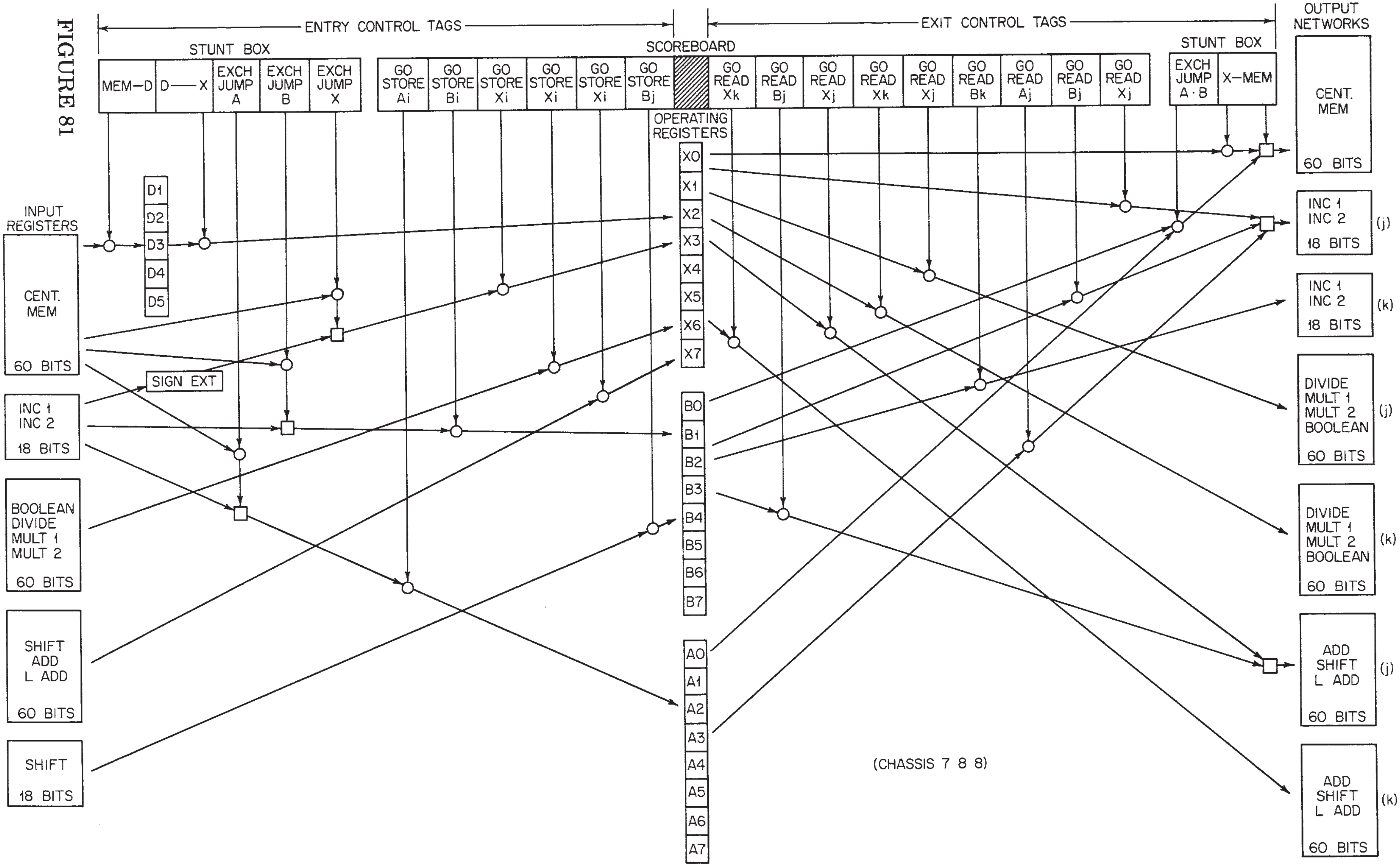
For the Increment Units, four Exit Control Tags are shown since they may specify an A, B, or X register with the j designator and only a B register with the k designator. Thus, the four tags, GO READ Xj, GO READ Bj, GO READ Aj, and GO READ Bk are used to gate data on this pair of trunks.

For the Multiply, Divide and Boolean trunk pair, only two controls are needed, GO READ Xj and GO READ Xk.

Three control tags are needed for the Add, Fixed Add, and Shift trunk-pair because of the use of Xj and/or Bk for Shift.

All GO READ and GO STORE tags are generated by the Scoreboard as described previously.

FIGURE 81


ENTRY CONTROL TAGS
STUNT BOX
MEM—D
D—X
EXCH JUMP A
EXCH JUMP B
EXCH JUMP X
GO STORE Ai
GO STORE Bi
GO STORE Xi
GO STORE Xi
GO STORE Xi
GO STORE Bj
SCOREBOARD
EXIT CONTROL TAGS
GO READ Xk
GO READ Bj
GO READ Xj
GO READ Xk
GO READ Xj
GO READ Bk
GO READ Aj
GO READ Bj
GO READ Xj
STUNT BOX
EXCH JUMP A·B
X—MEM
OUTPUT NETWORKS
INPUT REGISTERS
CENT. MEM 60 BITS
INC 1 INC 2 18 BITS
BOOLEAN DIVIDE MULT 1 MULT 2 60 BITS
SHIFT ADD L ADD 60 BITS
SHIFT 18 BITS
D1
D2
D3
D4
D5
SIGN EXT
OPERATING REGISTERS
X0
X1
X2
X3
X4
X5
X6
X7
B0
B1
B2
B3
B4
B5
B6
B7
A0
A1
A2
A3
A4
A5
A6
A7
CENT. MEM 60 BITS
INC 1 INC 2 18 BITS (j)
INC 1 INC 2 18 BITS (k)
DIVIDE MULT 1 MULT 2 BOOLEAN 60 BITS (j)
DIVIDE MULT 1 MULT 2 BOOLEAN 60 BITS (k)
ADD SHIFT L ADD 60 BITS (j)
ADD SHIFT L ADD 60 BITS (k)
(CHASSIS 7 8 8)

F. SUMMARY

It should be remembered at this point that this rather complex combination of translation and flag networks is intended to detect and cope with the first order, second order, and third order conflicts described previously. It should be obvious that an excessive amount of hardware for these networks would make the scheme worthless.

Actually, these networks, while complex, require less hardware than an average functional unit. To determine whether they are worth it, some examples are given below. Several simple rules are used in the timing of the examples, as follows.

1. Consecutive instruction words from storage require a minimum of eight minor cycles.
2. Double length instructions, 30 bits, require two minor cycles to issue.
3. A functional unit is free one minor cycle after the result is placed in a register.

The Appendix contains a comprehensive treatment of detailed timing considerations, of which the three above were picked to explain apparent anomalies.

EXAMPLE ONE

For a first example the solution to the following equation is timed.

$$AX^2 + BX + C = Y$$

The program to perform this solution is given below. The chart lists six times by minor cycle count. These are:

ISSUE — Relative time of instruction issue.
START — Start of function.
RESULT — Function complete with result available.
UNIT FREE — Unit ready for reuse.
FETCH — Operand fetched from storage and available in X Register.
STORE — Result stored in storage.

			ISSUE	START	RESULT	UNIT	FETCH	STORE
N1	$A1 = A1 + K1$	FETCH X	1	1	4	5	9	
	$A2 = A2 + K2$	FETCH A	3	3	6	7	11	
N2	$X0 = X1 * X1$	FORM X^2	9	9	19	20		
	$X6 = X0 * X2$	FORM AX^2	10	19	29	30		
	$A3 = A3 + K3$	FETCH B	11	11	14	15	19	
N3	$A4 = A4 + K_4$	FETCH C	17	17	20	21	25	
	$X3 = X3 * X1$	FORM BX	20	20	30	31		
	$X5 = X6 + X3$	FORM $AX^2 + BX$	21	30	34	35		
N4	$X7 = X5 + X4$	FORM Y	35	35	39	40		
	$A7 = A7 + K5$	STORE Y	36	39	42	43		47

If sequential times are counted with no losses due to instruction fetch, the list shown at the right would total 78 minor cycles, rather than the 46 shown. Example one is shown in Figure 82 in a different form.

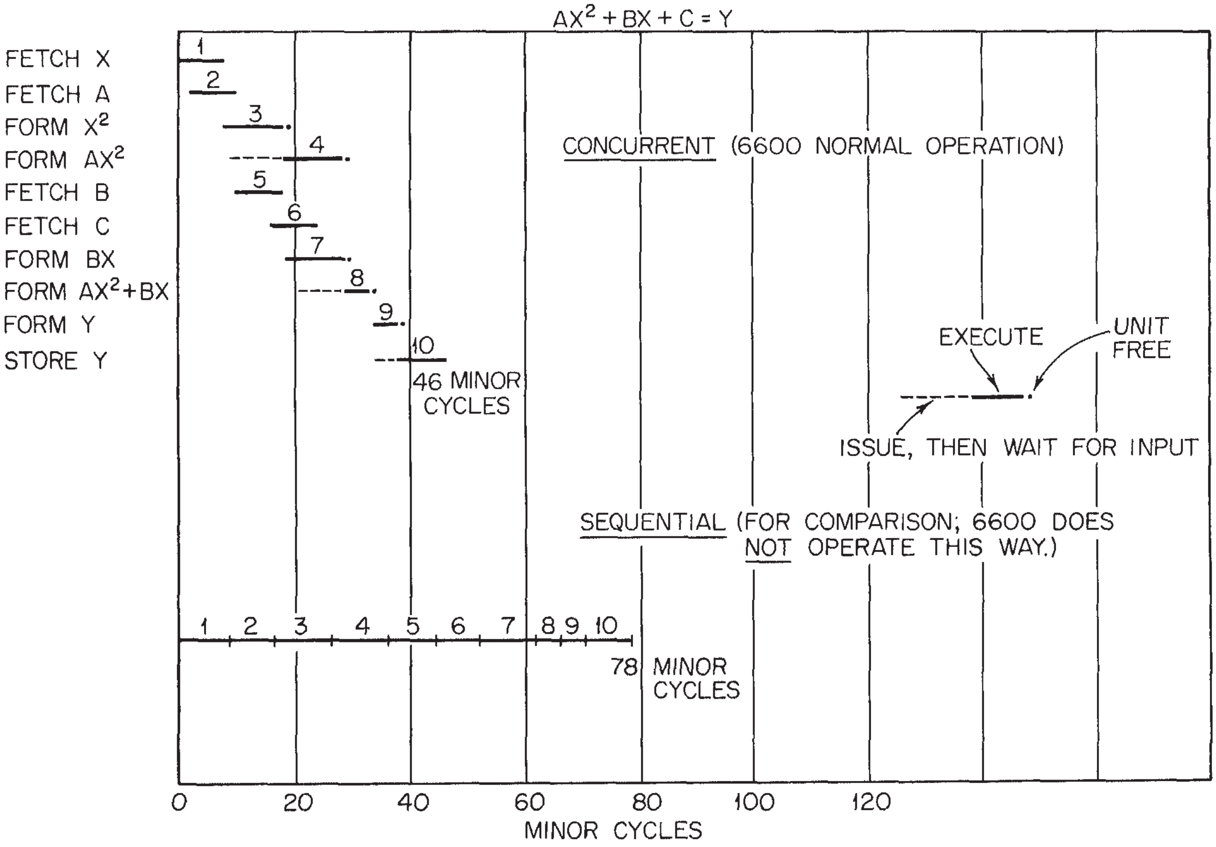


FIGURE 82

EXAMPLE TWO

The single solution shown in Example one can be repeated for a case in which both X and Y are considered vectors. A vector may be defined as a consecutive series of floating point numbers in central storage.

For this example the three values A, B, and C are considered constants. The solution can be conveniently described as an initial phase and a repetitive, or iterative, phase. The iterative phase can be held in the instruction stack for very high speed. The store address is presumed in register A7.

This totals 71 minor cycles, covering one initial pass, and 99 repetitions of 41 minor cycles each, using the instruction stack, for a total of 4,120 minor cycles. Without parallel functions, the initial phase is 107 minor cycles and 99 repetitions of 66 minor cycles each for a total of 6,641 minor cycles.

EXAMPLE THREE

An additional advantage can be obtained in Example two by making a slight modification of the "FORM Y" and "STORE Y" instructions. Specifi-

			ISSUE	START	RESULT	UNIT	FETCH	STORE
N1	A1 = A1 + K1	FETCH X	1	1	4	5	9	
	A2 = A2 + K2	FETCH A	3	3	6	7	11	
N2	A3 = A3 + K3	FETCH B	9	9	12	13	17	
	A4 = A4 + K4	FETCH C	11	11	14	15	19	
N3	B1 = B0 + 1	Set B1 to 1	17	17	20	21		
	B2 = B0 + K5	Set Vector Length	19	19	22	23		
		FIRST ITERATION						
N4	X0 = X1 * X1	FORM X^2	25	25	35	36		
	X6 = X0 * X2	FORM AX^2	26	35	45	46		
	X0 = X3 * X1	FORM BX	36	36	46	47		
	A1 = A1 + B1	FETCH NEXT X	37	37	40	41	45	
N5	B2 = B2 − B1	DECREMENT B2	41	41	44	45		
	X5 = X6 + X0	FORM $AX^2 + BX$	42	46	50	51		
	X7 = X5 + X4	FORM Y	51	51	55	56		
	A7 = A7 + B1	STORE Y	56	56	59	60	64	
N6	GO TO N4 IF B2 $\neq$ 0	BRANCH	60	—	72			
		SECOND ITERATION						
N4	X0 = X1 * X1	FORM X^2	72	72	82	83		
	N6 = X0 * X2	FORM AX^2	73	82	92	93		
	X0 = X3 * X1	FORM BX	83	83	93	94		
	A1 = A1 + B1	FETCH NEXT X	84	84	87	88	92	
N5	B2 = B2 − B1	DECREMENT B2	86	86	89	90		
	X5 = X6 + X0	FORM $AX^2 + BX$	87	93	97	98		
	X7 = X5 + X4	FORM Y	98	98	102	103		
	A7 = A7 + B1	STORE Y	103	103	106	107		111
N6	GO to N4 IF B2 $\neq$ 0	BRANCH	104	—	113			

cally, these two are positioned at the beginning of the iterative phase, as shown on page 140.

This optimization produces an initial phase of 55 minor cycles and 100 iterations of 26 minor cycles each for a total of 2,655 as against 6,641 minor cycles for the sequential equivalent. The last result can be accomplished by making 101 passes of the iterative phase or by adding the two required instructions in location N7. Note that there is an initial result which should be ignored.

These examples are not considered to be special cases. Highly efficient

			ISSUE	START	RESULT	UNIT	FETCH	STORE
N1	A1 = A1 + K1	FETCH X	1	1	4	5	9	
	A2 = A2 + K2	FETCH A	3	3	6	7	11	
N2	A3 = A3 + K3	FETCH B	9	9	12	13	17	
	A4 = A4 + K4	FETCH C	11	11	14	15	19	
N3	B1 = B0 + 1	SET B1 to 1	17	17	20	21		
	B2 = B0 + K5	Set Length	19	19	22	23		
		FIRST ITERATION						
N4	X7 = X5 + X4	FORM Y	25	25	29	30		
	X0 = X1 * X1	FORM X^2	26	26	36	37		
	X6 = X0 * X2	FORM AX^2	27	36	46	47		
	A7 = A7 + B1	STORE Y	30	30	33	34	—	38
N5	X0 = X3 * X1	FORM BX	37	37	47	48		
	A1 = A1 + B1	FETCH NEXT X	38	38	41	42	46	
	B2 = B2 − B1	DECREMENT B2	39	39	42	43		
	X5 = X6 + X0	FORM $AX^2 + BX$	40	47	51	52		
N6	GO TO N4 IF B2 $\neq$ 0	BRANCH	44			56		
		SECOND ITERATION						
N4	X7 = X5 + X4	FORM Y	56	56	60	61		
	X0 = X1 * X1	FORM X^2	57	57	67	68		
	X6 = X0 * X2	FORM AX^2	58	67	77	78		
	A7 = A7 + B1	STORE Y	61	61	64	65		69
N5	X0 = X3 * X1	FORM BX	68	68	78	79		
	A1 = A1 + B1	FETCH NEXT X	69	69	72	73	77	
	B2 = B2 − B1	DECREMENT B2	70	70	73	74		
	X5 = X6 + X0	FORM $AX^2 + BX$	71	78	82	83		
N6	GO TO N4 IF B2 $\neq$ 0	BRANCH	73		82			

use of the instruction stack is perhaps unusual in programs generated by compilers such as FORTRAN. However, the presence of such a powerful mechanism also offers incentive for extending the compiler to take advantage of it. As you would expect, 6600 compilers have shown a steady improvement as more concurrency is introduced. In any event, the concurrency of functional unit operation is evident even without optimization.

PERIPHERAL SUBSYSTEM

VII

As described in Chapter II, the Peripheral Subsystem is made up of ten small processors and twelve standard channels. In following sections, these will be described, together with several key peripheral devices.

A. PERIPHERAL PROCESSORS

The processing requirements in an input-output section of any computer include the following.

- Transferring data between peripheral device and central storage.
- Controlling the initiation of peripheral device actions.
- Establishing priorities between devices.
- Buffering data between asynchronous devices.
- Interrupting the central processor for execution of priority tasks.

This list is not exhaustive but illustrates the nature of the processing assigned to the ten small processors in the 6600 Computer.

These Peripheral and Control Processors, or PPU's, are constructed within the main frame cabinet of the 6600. This provides convenient use of identical logic and storage modules as in the central processor and central storage plus the shared use of the power and cooling system. It also allows a unique form of design, called the "barrel," which is shown in Figure 83.

Instead of independently constructed PPU's the barrel design utilizes a

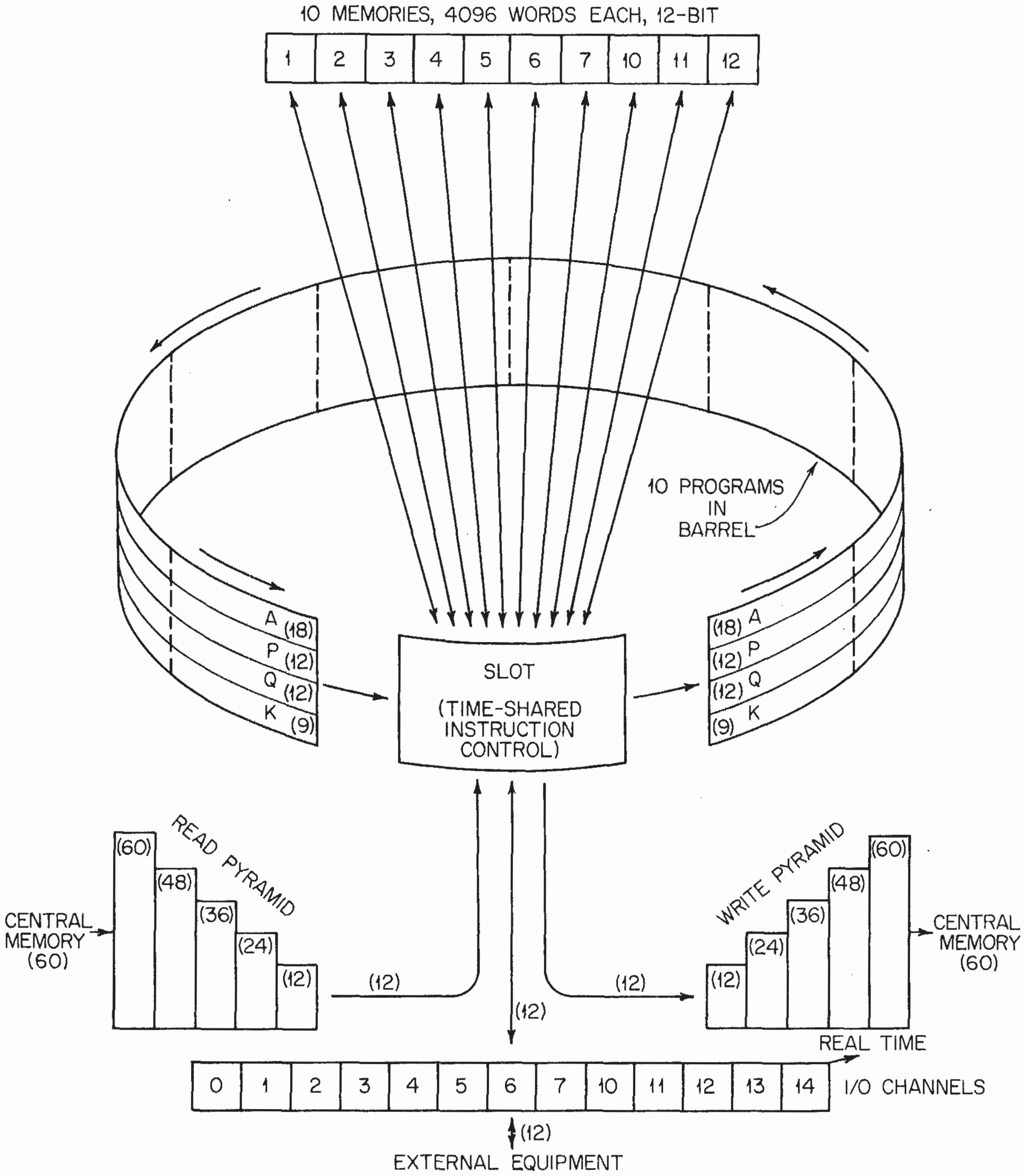

10 MEMORIES, 4096 WORDS EACH, 12-BIT
1
2
3
4
5
6
7
10
11
12
10 PROGRAMS IN BARREL
A (18)
P (12)
Q (12)
K (9)
SLOT
(TIME-SHARED INSTRUCTION CONTROL)
READ PYRAMID
WRITE PYRAMID
(60)
(48)
(36)
(24)
(12)
CENTRAL MEMORY (60)
REAL TIME
I/O CHANNELS
0
1
2
3
4
5
6
7
10
11
12
13
14
EXTERNAL EQUIPMENT

FIGURE 83 Peripheral processors.

network of registers to share one common arithmetic, logical, and distribution system. The barrel contains, logically, ten positions, each one representing a PPU. One position is labeled the "slot," in which one step can be performed. Typically, a PPU instruction requires several steps for execution. Each step is a comfortable fit of the storage cycle, 1000 nanoseconds and the arithmetic, logical or data transfer cycle required, a minor cycle of 100 nanoseconds. For example, the sum of an operand from the PPU storage and the PPU A register requires only 100 nanoseconds for the arithmetic but 1000 nanoseconds for the operand storage reference. This convenient "fit" is emphasized by the choice of *ten* PPU's time-sharing the common slot.

Once every minor cycle, 100 nanoseconds, all information in the barrel is moved one position. The information for one PPU is therefore moved through the slot position once each major cycle. All ten PPU's are time shared in this manner by the slot hardware, without degrading their performance.

INSTRUCTION FORMATS

Two formats are used in the PPU as shown in Figure 84.

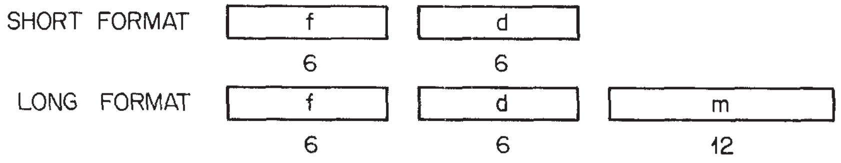


FIGURE 84

The short format is held in one PPU storage location, whereas the long format requires two consecutive PPU storage locations. These two formats allow a very flexible operand addressing scheme.

A particularly useful property is the assignment of the first 64 PPU storage locations. These locations can be directly addressed by the d field of the short format. This is a fast operation and also uses only one storage location for the instruction. These locations can be conveniently used for temporary storage, pointers, tables, and so on.

Other addressing combinations are described in the following list.

d	Implies d itself.
(d)	The contents of address d, one of the 64 initial storage locations.
((d))	The contents of the location specified by (d).
m	Implies m itself used as an address.
m + (d)	The contents of d are added to m to form a jump address.
(m + (d))	The contents of d are added to m to form the address of an operand.
dm	An 18-bit quantity with d as the upper six bits and m as the lower twelve bits.

JUMPS

The first set of instructions to be described are the Jump instructions. These instructions provide for conditional branches, with the destination relative to the current program address, and unconditional branches, with the destination formed by a base address plus index.

		Time (Major Cycles)
00	Pass	1
01	Long Jump to m + (d)	3
02	Return Jump to m + (d)	4
03	Unconditional Jump d	1
04	Zero Jump d	1
05	Nonzero Jump d	1
06	Plus Jump d	1
07	Minus Jump d	1

Long Jump, 01, and Return Jump, 02, utilize the m field of the long format as a base address and one of the first 64 storage locations as an index. Return Jump assumes that a long jump is stored at the destination address. Program address of the next instruction following the Return Jump, (P) + 2, is placed in the m field of that assumed instruction. Program control is then transferred to the next location following the assumed Long Jump instruction. Typical usage of this instruction will be as a "normal exit" from the program entered by Return Jump, as shown in Figure 85.

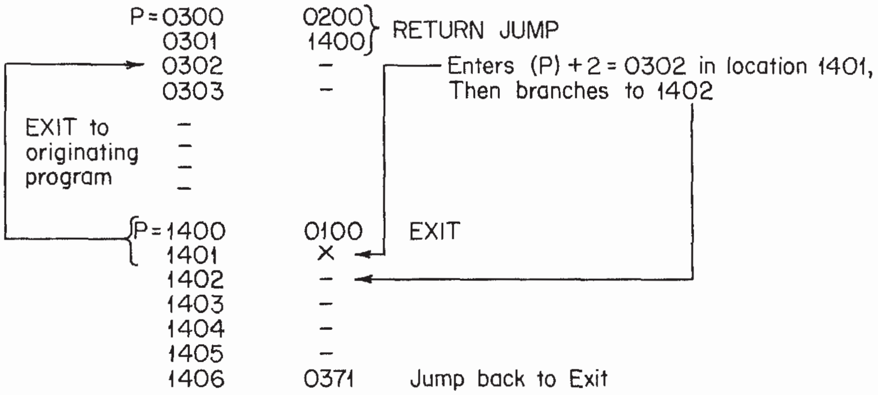

P = 0300
0301
0302
0303
0200
1400
RETURN JUMP
Enters (P) + 2 = 0302 in location 1401, Then branches to 1402
EXIT to originating program
P = 1400
1401
1402
1403
1404
1405
1406
0100
X
EXIT
0371
Jump back to Exit

FIGURE 85

Unconditional Jump d and the four Condition Jumps utilize the current program address (P) as a base and the d field of the instruction as a signed relative index. If the d field is positive, the effect is to branch forward by the amount of d. If the d field is negative, the branch is backward. The example in Figure 85, in which an Unconditional Jump, 0371, branches back to the "Exit," shows a negative d field, 71. This is the equivalent of minus 6 octal, causing the program address P to be reduced from 1406 to 1400.

The Conditional Jumps are used to test the current condition of the A Register and are self-explanatory.

NO ADDRESS

A set of instructions, classified as No Address, utilizes the d field or the dm field as constants. In these instructions no additional storage references are required beyond those needed to obtain the instruction itself.

		Time (Major Cycles)
10	Shift d	1
11	Logical Difference d	1
12	Logical Product d	1
13	Selective Clear d	1
14	Load d	1
15	Load Complement d	1
16	Add d	1
17	Subtract d	1
20	Load dm	2
21	Add dm	2
22	Logical Product dm	2
23	Logical Difference dm	2

An implied destination, the A Register, receives the results of the above operations. In some cases, as described below, the A Register is also an input operand for the operation.

10 Shift d
This instruction shifts the contents of A right or left d places. If d is positive $(00\text{-}37)_8$, the shift is left circular; if d is negative $(40\text{-}77)_8$, A is shifted right open-ended without sign extension.

11 Logical Difference d
This instruction forms in A the bit-by-bit logical difference, or exclusive OR, of d and the lower six bits of A, leaving the upper twelve bits of A unaltered.

12 Logical Product d
This instruction forms the bit-by-bit logical product, or AND, of d and the lower six bits of A, leaving zeroes in the upper twelve bits of A.

13 Selective Clear d
This instruction clears any of the lower six bits of A where there are corresponding ones in d, leaving the upper twelve bits of A unaltered.

14 Load d
Thus instruction clears the A register and loads d.

15 Load Complement d
This instruction clears the A register and loads the complement of d, sign extended.

16 Add d
This instruction adds d (treated as a six-bit positive quantity) to the contents of the A register.

17 Subtract d
This instruction subtracts d (treated as a six-bit positive quantity) from the contents of A.

Only one major cycle is needed for the above instructions to fetch the instruction itself since no further storage references are needed. This, of course, means that the entire operation is accomplished in the one minor cycle slot time following the instruction fetch.

The following NO ADDRESS instructions require two major cycles to complete the fetch of the long-format instructions.

20 Load dm
This instruction clears A and loads an 18-bit quantity consisting of d as the higher six bits and m as the lower twelve bits.

21 Add dm
This instruction adds to A the 18-bit quantity dm.

22 Logical Product dm
This instruction forms in A the bit-by-bit logical product, or AND, of the contents of A and the 18-bit quantity dm.

23 Logical Difference dm
This instruction forms in A the bit-by-bit logical difference, or exclusive OR, of the contents of A and the 18-bit quantity dm.

DIRECT-INDIRECT-INDEX

A set of instructions is included in the PPU's which allow addressing by direct, indirect, or indexed modes. Direct mode means d is used as the address of PPU storage, specifying one of the first 64 storage locations. Indirect mode means the contents of the storage location, specified by d, are used to specify the storage location of the operand. Index mode means that the m field of the instruction serves as the base address of the operand, to be modified by (d). If $d = 0$, the operand address is simply m; but if $d \neq 0$, then $m + (d)$ is the operand address.

DIRECT

		Time (Major Cycle)
30	Load (d)	2
31	Add (d)	2
32	Subtract (d)	2
33	Logical Difference (d)	2
34	Store (d)	2
35	Replace Add (d)	4
36	Replace Add One (d)	4
37	Replace Subtract One (d)	4

INDIRECT

40	Load ((d))	3
41	Add ((d))	3
42	Subtract ((d))	3
43	Logical Difference ((d))	3
44	Store ((d))	3
45	Replace Add ((d))	5
46	Replace Add One ((d))	5
47	Replace Subtract One ((d))	5

INDEX

50	Load (m + (d))	3-4
51	Add (m + (d))	3-4
52	Subtract (m + (d))	3-4
53	Logical Difference (m + (d))	3-4
54	Store (m + (d))	3-4
55	Replace Add (m + (d))	5-6
56	Replace Add one (m + (d))	5-6
57	Replace Subtract one (m + (d))	5-6

For simplicity, these instructions are described in groups of three to show the three addressing options. Note above that each additional PPU storage reference to accomplish the indirect or index simply requires an additional Major Cycle. When d = 0 in the index case, no additional Major Cycle is needed.

30 Load (d)
40 Load ((d))
50 Load (m + (d))

These instructions clear the A Register and load the twelve-bit quantity from storage, leaving the upper six bits of A zero.

31 Add (d)
41 Add ((d))
51 Add (m + (d))

These instructions add to the A Register the twelve-bit operand from storage, treated as a twelve-bit positive quantity.

32 Subtract (d)
42 Subtract ((d))
52 Subtract (m + (d))

These instructions subtract from the A Register the twelve-bit operand from storage, treated as a twelve-bit positive quantity.

33 Logical Difference (d)
43 Logical Difference ((d))
53 Logical Difference (m + (d))

These instructions form in A the bit-by-bit logical difference, or exclusive OR, of the lower twelve bits of A and the twelve-bit operand from storage, leaving the upper six bits of A zero.

34 Store (d)
44 Store ((d))
54 Store (m + (d))

These instructions store the lower twelve bits of A in the specified PPU storage location.

35 Replace Add (d)
45 Replace Add ((d))
55 Replace Add (m + (d))

These instructions add the quantity from storage to A and store the lower twelve bits of the result at the same storage location. The resultant sum is left in A.

36 Replace Add One (d)
46 Replace Add One ((d))
56 Replace Add One (m + (d))

These instructions replace the quantity in the storage location with its initial value plus one. The resultant sum is left in A, destroying the previous contents of A.

37 Replace Subtract One (d)
47 Replace Subtract One ((d))
57 Replace Subtract One (m + (d))

These instructions replace the quantity in the storage location with its initial value minus one. The resultant difference is left in A, destroying the previous contents of A.

CENTRAL PROCESSOR AND CENTRAL STORAGE

Instructions are included in the PPU repertoire which allow each PPU to cause an Exchange Jump in the CPU and also to monitor the CPU program address.

2600 Exchange Jump

This instruction transmits the 18-bit quantity in the A register to the Exchange Jump mechanism of the CPU with an initiating signal. As described in Chapter VI, the Central Processor is interrupted; an exchange is made between the CPU Registers and Exchange Package in Central Storage at the location obtained from the PPU A Register; and finally the CPU is started on the new program.

261j Monitor Exchange Jump (Optional)

This instruction is provided as an *option* in conjunction with the Central Exchange Jump, CEJ. The d field of this short-format instruction is split into two octal digits, including the option designator (1) and a CPU designator (j), for use with the two-CPU 6500 Computer. Monitor Exchange Jump operates exactly the same as EXN, Exchange Jump above, only if the Monitor Flag, in the CPU, is cleared. The flag is then set, indicating that the CPU is in the monitor state. If the flag is set, this instruction is a PASS. Therefore, a confirmation routine is required involving a simple communication between the CPU monitor and the PPU.

272 Read Program Address
This instruction transfers the contents of the CPU program address register to the PPU A Register to allow PPU monitoring of the condition of the CPU program.

Each PPU can access the Central Storage by single word or block transfer, using the following instructions.

60 Central Read from (A) to d
This instruction transfers a 60-bit word from central storage to five consecutive locations in the PPU storage. The 18-bit address of central storage must be loaded in the PPU A Register prior to this instruction. The five twelve-bit portions of the 60-bit word are disassembled from left to right and loaded consecutively beginning at PPU location d.

61 Central Read (d) words from (A) to m
This instruction provides a block transfer from Central Storage to PPU Storage. The 18-bit address of the beginning word in Central Storage must be loaded in the PPU A Register prior to this instruction.

During this block transfer, the PPU program address is temporarily placed in PPU storage location 0 in order that the program address increment mechanism can be used to advance m. The PPU Q Register is used to decrement the contents of location d.

The block of central storage locations goes from address (A) to address (A) + (d) − 1. The block of PPU storage locations goes from address m to m + 5(d) − 1. See Figure 86.

62 Central Write to (A) from d
This instruction assembles five consecutive twelve-bit words from PPU storage into a 60-bit word and stores the word in Central Storage. The 18-bit address of Central Storage must be loaded in the PPU A Register prior to this instruction. The first twelve-bit word appears as the left-most, or higher order, portion of the 60-bit word.

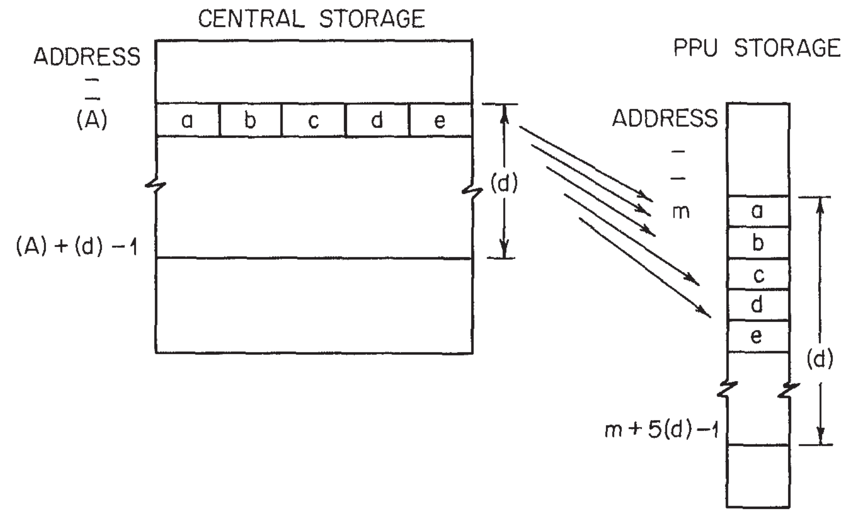


FIGURE 86

63 Central Write (d) words from m to (A)
This instruction assembles a block of 60-bit words and writes them in Central Storage. The mechanics of the execution are identical with Central Read above with the exception of the direction of data flow.

INPUT/OUTPUT

All PPU's have access to the twelve I/0 channels in turn during their portion of time in the slot. At this time, data may be transferred or conditions sampled.

Two flags are utilized for each channel in order to control the channel and to indicate its status.

Active/Inactive Flag
Each channel has this flag to indicate that it has been selected for use and is busy.

Full/Empty Flag
Each channel has this flag to indicate that the channel register contains a word.

Each channel contains a register used for either direction of data flow. Data may pass between PPU's through these channels if desired, using the PPU instructions.

The following instructions are provided for sampling channel conditions.

64 Jump to m if channel d active
65 Jump to m if channel d inactive
These instructions transfer the program sequence to storage location m if the condition of the active/inactive flag for channel

d is "true." Otherwise, the current program sequence is continued.

66 Jump to m if channel d full
67 Jump to m if channel d empty

These instructions transfer the program sequence to storage location m, if the condition of the full/empty flag for channel d is "true." Otherwise, the current program sequence is continued.

Data transfer on the channels is controlled by instructions which provide single word transfer or block transfer.

70 Input to A from channel d
72 Output (A) on channel d

These instructions transfer a word between the A Register and channel d.

71 Input (A) words to m from channel d
73 Output (A) words from m on channel d

These instructions transfer a block of words between the PPU Storage and channel d. Similar to the Central Read and Central Write, the program address is temporarily stored in PPU storage location 0 so that the program address increment mechanism can be used to increment m. The content of A is decremented to control the length of the block transfer.

Control over the channel is provided by two PPU instructions.

74 Activate channel d

This instruction activates the channel specified by d. This sets the active flag for channel d and also signals "active" on the channel to the I/O equipment connected.

75 Disconnect channel d

This instruction deactivates the channel specified by d. This clears the active flag for channel d and also signals "inactive" on the channel to the I/O equipment.

Control of the I/O equipment connected to a channel is provided by two instructions.

76 Function (A) on channel d
77 Function m on channel d

These instructions transfer a twelve-bit word on channel d, either from A or the m instruction field, together with a "function" signal.

Typical reaction to such functions is the setting or clearing of control switches in the I/O equipment.

Channel controls are very simple as can be seen in Figure 87.

The two flags are set and cleared as shown both from the PPU and from the channel equipment.

Included within the PPU logic for convenience is a Real Time clock which is available to all PPU's on peripheral channel 12. This clock "ticks" every major cycle, or one microsecond. The clock is, in fact, a twelve-bit counter which sweeps, or "starts over," every four milliseconds. The operating system can utilize this mechanism to construct a "day" clock or other timing counts.

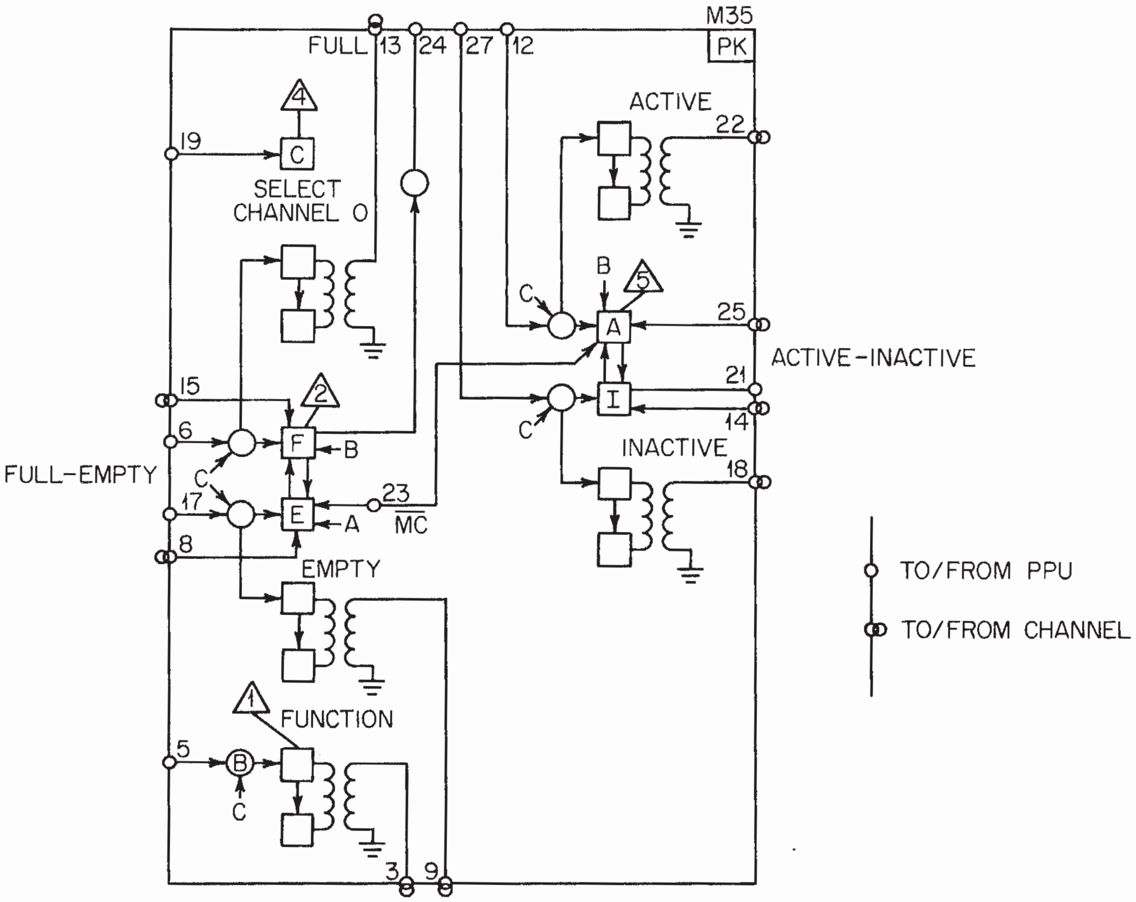

M35
PK
FULL
13
24
27
12
4
19
C
SELECT CHANNEL 0
ACTIVE
22
B
5
C
A
25
ACTIVE-INACTIVE
21
I
14
C
INACTIVE
18
15
2
6
F
B
FULL-EMPTY
C
17
E
A
23
$\overline{MC}$
8
EMPTY
TO/FROM PPU
TO/FROM CHANNEL
1
FUNCTION
5
B
C
3
9

FIGURE 87

BARREL

Four registers are contained in each position of the PPU barrel. These are:

• A Register	18 bits
• P Register	12 bits
• Q Register	12 bits
• K Register	9 bits

The first two, A and P are explicitly defined and referenced in the PPU instructions. The other two, Q and K, are temporary holding registers providing for various operations. The Q Register, for example, holds:

- The address of the operand during direct addressing,
- The address of the address of the operand on indirect addressing,
- The peripheral address of data used during central read or write instructions,
- The upper six bits during constant mode instructions,
- The channel number on all I/O instructions and channel jump instructions,
- The shift count on shift instructions,
- The specific number of locations to jump on relative jumps.

The K Register holds the six-bit function code, F, of the current instruction, and a count of the number of major cycles taken.

Short-format, no-address instructions do not use the K Register since the translation is performed directly on the instruction and the execution is completed in one cycle.

The above values completely define the "state" of each PPU for use in the next slot time. When these data reach the slot in their turn, one instruction step can be completed.

Figure 88 is a diagram of the slot showing the major inputs, outputs, and functions performed.

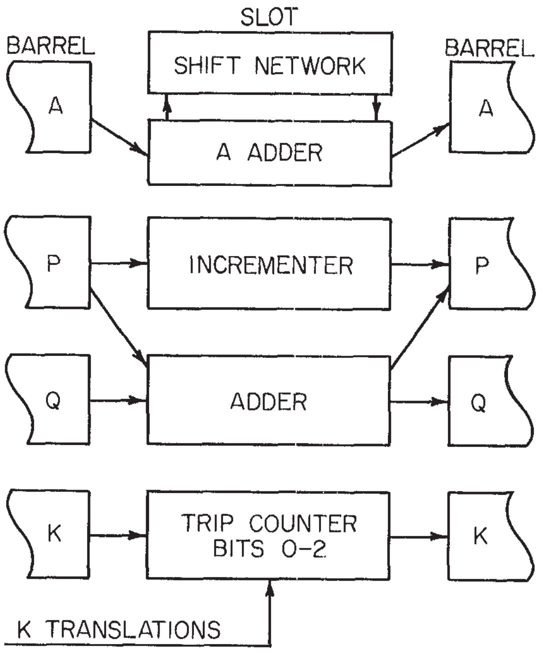


FIGURE 88 Elements of the slot.

A storage address for a particular processor is taken directly from the barrel six minor cycles before that processor is ready to enter the slot. This allows time for operands to be obtained from storage to be used in the slot. Translation of K is also begun in advance of the slot in order to control the slot operation.

The A Adder is used to execute add, subtract, selective clear, logical product, and logical difference instructions.

The Shift network is similar to that of the CPU, Chapter V. The shift is completed in parallel in one pass through the slot.

The P Incrementer is able to add zero or one to P. In instructions which require several trips around the barrel, P is, of course, incremented only once.

The Q Adder is used to compute relative addresses, indexed addresses, and to provide connective paths between P and Q.

The K counter produces a trip count which controls the sequence of operations for each instruction. This count can be separately set to handle repetitive sequences, for example, for block transfers.

CENTRAL READ/WRITE PYRAMIDS

Assembly and disassembly of twelve-bit and 60-bit words is accomplished in two pyramid networks. During Central Read operations, a 60-bit word enters the Read pyramid. In subsequent major cycles the PPU which initiated the operation removes twelve-bit words left to right; the process actually causes the remaining bits to move through the pyramid, a row at a time. This opens up the pyramid to another 60-bit entry. Since only one Central Storage access is allowable from all PPU's at one time, the words move through the pyramid in step.

A similar operation is performed in the Write pyramid, with each PPU entering at the correct point depending upon the number of assembly cycles it has taken. Again, another PPU may make use of the pyramid and will keep in step.

B. DEAD START

A typical obstacle to the understanding of a complex instrument is the inability to discover the answer to the question, "How does it start?" This section is devoted to answering that question.

A first assumption is made that there are a number of peripheral devices capable of loading the computer's operating system. These may be:

- Magnetic tape
- Punched cards
- Magnetic disk

A second assumption is made that a direct entry of "machine language" programs is possible from these devices.

To activate the DEAD START sequence, the 6600 cabinet contains a panel containing several control switches and a 12 × 12 matrix of switches. See Figure 89. Also contained are switches for performing maintenance tests.

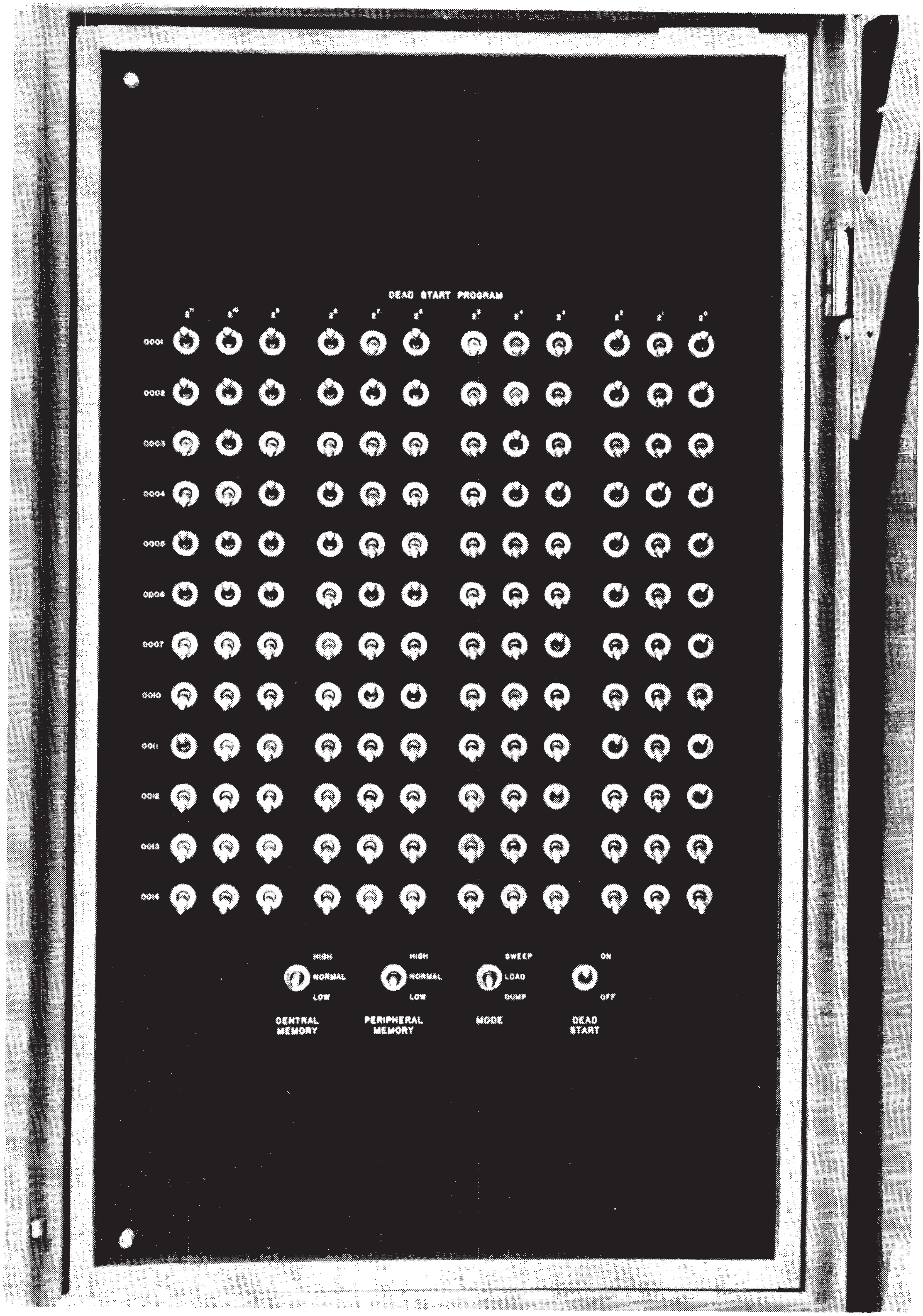

DEAD START PROGRAM
2^{11}
2^{10}
2^9
2^8
2^7
2^6
2^5
2^4
2^3
2^2
2^1
2^0
0001
0002
0003
0004
0005
0006
0007
0010
0011
0012
0013
0014
HIGH
NORMAL
LOW
CENTRAL MEMORY
HIGH
NORMAL
LOW
PERIPHERAL MEMORY
SWEEP
LOAD
DUMP
MODE
ON
OFF
DEAD START

FIGURE 89

In order to load an initial system program, two control switches are activated. The SWEEP/LOAD/DUMP switch is set to the LOAD position. A very simple "program" is then set up in the 12 × 12 matrix. Finally, the DEAD START switch is turned on momentarily, then off.

While the DEAD START switch is in the "ON" position, a MASTER CLEAR/DEAD START signal is repetitively transmitted throughout the system. This is a one-microsecond signal and is transmitted every 4096 microseconds, until the DEAD START is turned off. This signal prepares the entire system for start by presetting and clearing flip-flops throughout the system logic. The signal also:

- Assigns each PPU to an I/O channel corresponding to its number; for example, PPU 0 to channel 0, PPU 1 to channel 1, etc.
- Sets all I/O channels to Active and Empty.
- Sets all PPU's to an intermediate step of an INPUT instruction, waiting data on the I/O channel.
- Transmits a MASTER CLEAR to the peripheral equipment on each channel.
- Sets all PPU's to program address 0.
- Sets location 0000 in all PPU's to zero.
- Sets A for all PPU's to an input word count of 10,000.
- Sets the CPU to STOP.

This pulse is originally generated from the Real Time clock which is a twelve-bit counter connected directly to the Major Cycle.

Following this preparation, the DEAD START Synchronizer connected to I/O Channel 0 is activated. First action is a "Full" pulse on Channel 0 with no data. PPU 0 receives the "Full" signal, stores the zeroes from Channel 0 input register in location 0000, and sends an Empty pulse to the DEAD START Synchronizer. The DEAD START Synchronizer transmits the twelve words from the 12 × 12 matrix of the DEAD START Panel on Channel 0 to be stored in locations 0001 through 0014 (octal) of PPU 0. Following the last word, the DEAD START Synchronizer sends a disconnect on Channel 0 which causes PPU 0 to exit from the Input instruction.

The exit from an INPUT instruction, as described previously, involves recovering the program address from location 0, incrementing it by one, and beginning the program sequence at the resultant location. Since the contents of address 0000 is zero, the initial program address is 0001. In other words, PPU 0 is caused to begin at the first word loaded from the DEAD START panel.

The twelve-word "program" thus loaded can be a considerable aid in very basic maintenance of the computer. To load the normal operating system, however, a program such as the following can be used.

PROGRAM

This program selects a magnetic tape unit for input, then waits for the tape unit to be manually activated. The typical initial program on the

TABLE VI Dead Start Panel Settings

(Bootstrap Loading of the System Tape for 6000 Series Tape Units Only)

Memory*	Contents	Action Generated	Toggle Settings
01	1410	Load (A) with 10_8	001 100 001 000
02	730x	Output 10_8 words starting at location 6 on channel x (processor x will store these in its memory beginning at location 0)	111 011 000 xxx
03	0006		000 000 000 110
04	750x	Disconnect channel x (permits execution of program)	111 101 000 xxx
05	7113	Set to input mode (7770 words to location 0000 on channel 13)	111 001 001 011
06	0000		000 000 000 000
07	770x	Select rewind tape on channel x	111 111 000 xxx
10	2060		010 000 110 000
11	770x	Read up to 10,000 words in binary mode on channel x	111 111 000 xxx
12	2020		001 000 010 000
13	740x	Activate channel	111 100 000 xxx
14	710x	Set to input mode (channel x)	111 001 000 xxx
15	0000	Cleared during dead start	

* Locations at peripheral processor 0.

magnetic tape is sufficient to "bootstrap" the remainder of the operating system.

As this bootstrap program is entered, the remaining nine PPU's are loaded through PPU 0. To accomplish this, PPU 0 loads each channel acting as an input unit. When each PPU contains a system "resident" program, PPU 0 disconnects each channel, thereby causing each PPU to begin the "resident" program sequence. In the course of loading each PPU, PPU 0 can also set location 0000 for each to a number other than zero. This will cause the PPU to begin at a location in the loaded program other than address 0001 and can be convenient. The CPU is, of course, started by an Exchange Jump as described previously.

C. DISK STORAGE

Among the many devices which may be connected to a 6600 input-output channel, an essential secondary storage unit is the 6638 Magnetic Disk Storage. This unit provides about 800 million bits of on-line storage in one cabinet (Figure 90).

Storage is accomplished by magnetically recording on many flat disk surfaces. The read-write heads can be positioned to a number of tracks with

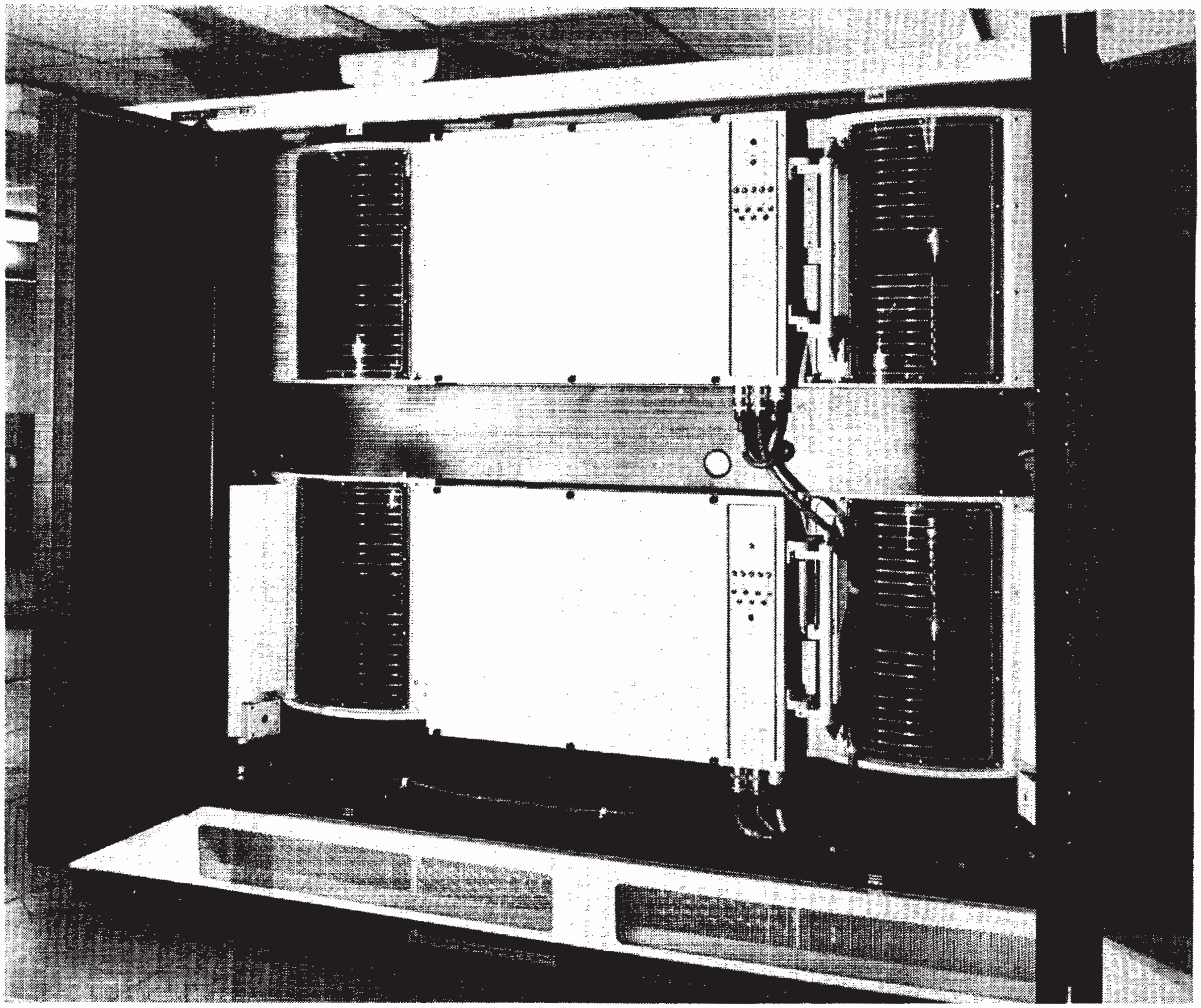


FIGURE 90

a hydraulic mechanism. The heads are maintained at a very close spacing with the disk surface by means of an air bearing formed as a result of the surface shape of the head and the spinning disk.

Disks are grouped in four "quadrants" using two vertically mounted spindles. See Figure 91.

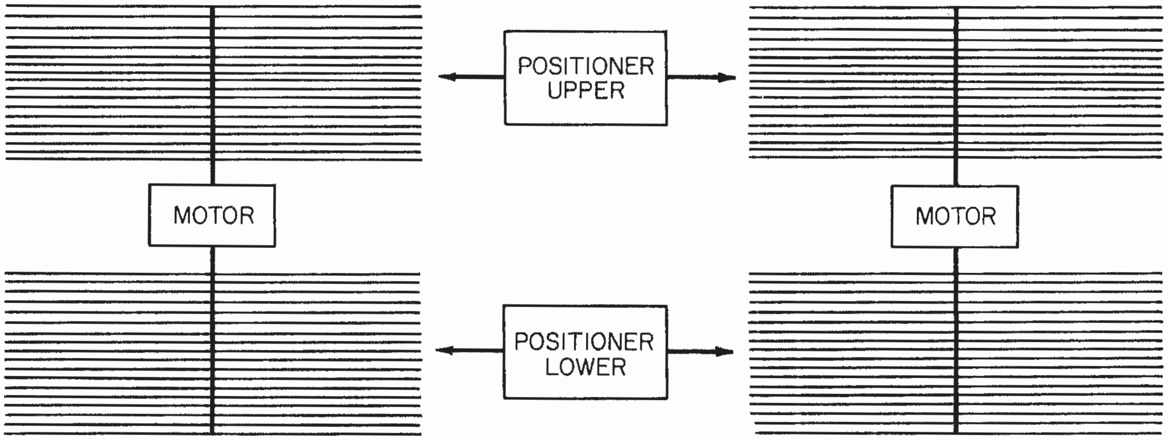

POSITIONER
UPPER
MOTOR
MOTOR
POSITIONER
LOWER

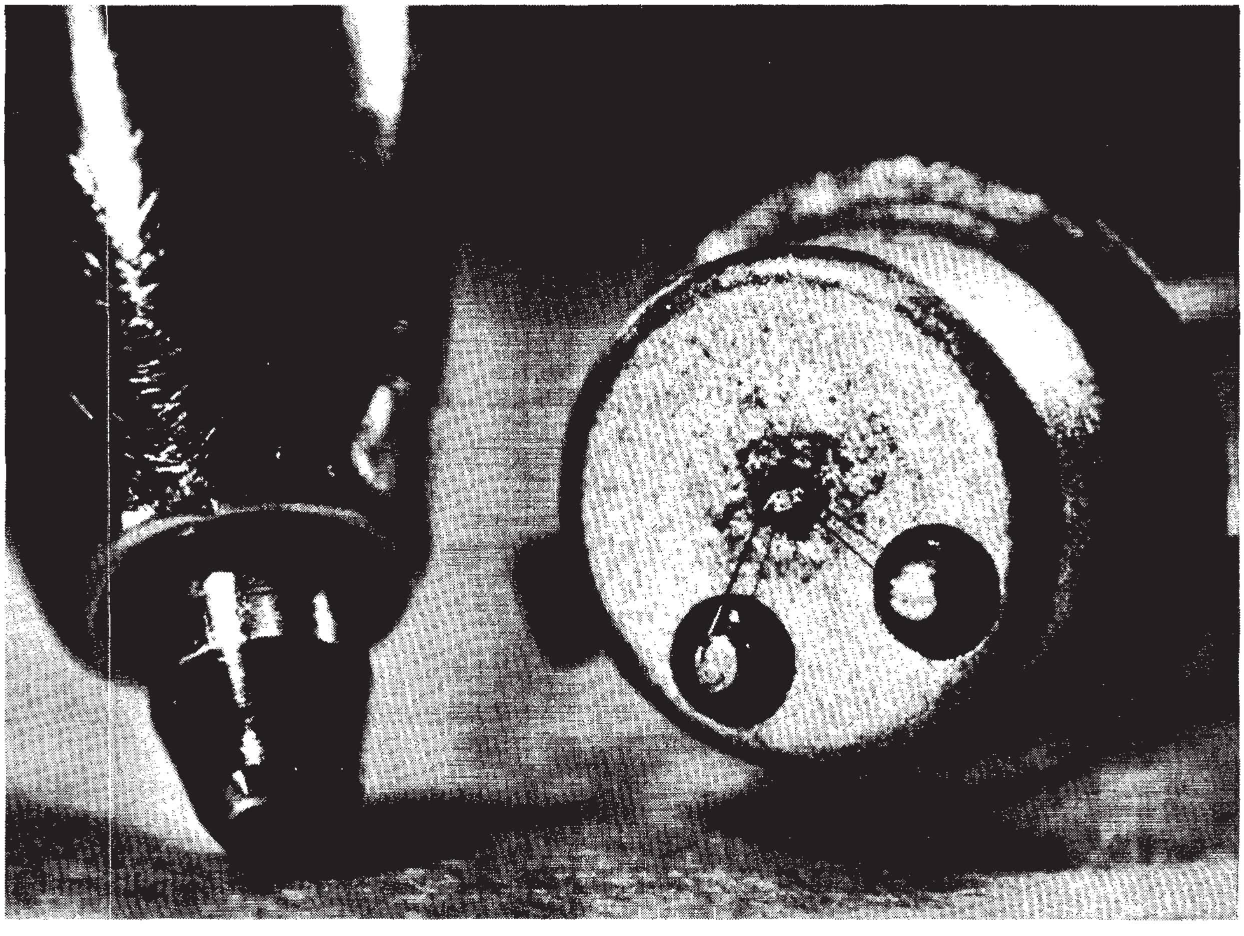
FIGURE 91

Motors are mounted between each vertical quadrant on each spindle, directly driving each at about 1200 revolutions per minute. Each quadrant contains eighteen disks with 32 data surfaces. Heads are mounted on arms connected to two "reactive" positioners, one for the upper pair of quadrants, and one for the lower pair. When positioning to a new track, the hydraulic mechanism causes all heads in the appropriate half of the cabinet to move; that is, upper or lower. Note that the movement of the head assembly in the left disk quadrant is counteracted by an opposite movement of the head assembly in the right disk quadrant.

For convenience, the upper pair of quadrants can be considered separate and independent of the lower pair. The 6638 Controller thus has the ability to operate each half as separate disk units. The controller also allows connection to two independent I/O channels. These may be on separate 6600 Computers, thereby providing access for both to common secondary storage.

Data is stored in the 6638 Disk Storage in fixed length blocks of 64 central storage words of sixty-bit length. Each track contains 100 sectors of 322 cells per sector with gaps between each sector to provide control space for headers and to provide ability to alter sectors.

Twelve heads are utilized in parallel on a read or write operation. Three heads from each of four head pads are, therefore, operating together. (Note:



Transistor mounted on header with leads bonded to posts. Ball-point pen at left.

There are six heads mounted in each head pad having access to one disk surface.) There are, therefore, 322 twelve-bit "bytes" transferred in one sector read or write. This is equivalent to a block of 64, 60-bit, central storage words with two extra bytes for control. This block, or data sector, is stored in the PPU controlling the Disk Storage, taking up 322 PPU storage locations.

In long transfers, alternate sectors are transferred by the PPU. In this case, the PPU reads from the Disk into PPU storage, then block transfers to Central Storage. The PPU can then return to the Disk in time for the next alternate sector. Because a disk surface can contain a small number of flaws, it is convenient for the PPU to keep track of these flaws by "half-tracks." These half-tracks refer to the two sets of alternate sectors present in one head position.

Data files are also allocated within the disk storage unit on a half-track basis. This reduces the allocation work load on the PPU and tends to encourage longer transfers between disk and central storage. This last is a definite value in efficiency.

Positioning of the head arms provides access to 192 tracks per arm. Since there are six heads in one head pad mounted on each arm, this requires 32 actual positions of the arm. These characteristics are listed below for convenient reference.

1. 72 Disks, 64 used for data.
2. 128 surfaces for data, with six heads per surface.
3. 12 heads parallel.
4. Positioning time—25 milliseconds minimum to 150 milliseconds maximum.
5. Latency (time for one revolution)—52 milliseconds.
6. Sector size—322 cells per track.
7. Sector gap—108 cells.
8. Sectors per revolution—100.
9. Two positioners, 32 positions each.
10. Data per position, each positioner—12.4 million bits.
11. Capacity total—792 million bits (for this sector size).

A normal sequence of control by a PPU over the Disk Storage is as follows.

1. Connect and Status
   This control step is initiated by a "Function" instruction in the PPU. The function code is transmitted to the Disk Storage Controller over the peripheral channel and attempts to connect the unit to the channel. If the other access channel already has control of the unit, the connect is not achieved.

   If connection is made, subsequent functions and data transfers can be performed. If connection is not made, only status can be read.

2. Activate Channel
   This control step is initiated by the PPU instruction "Activate Channel d" which causes the Disk Storage Controller to present status information on the input channel. A request status function may also be inserted here with more complex configurations on the channel.

   Status information includes:
   - Current sector address
   - Parity error
   - Not ready
   - Not connected
   - Lost data

3. Select Position
   This PPU "Function" instruction causes the positioning mechanism to seek one of 32 positions.

4. Select Head Group
   This PPU "Function" instruction causes the selection of one of 32 head groups. Each head group contains twelve heads for twelve-bit parallel operation.

5. Read (or Write)
   This step is initiated by a PPU block transfer, either input or output, of one sector. Because Disk Storage requires a minimum of one sector for read or write, the block transfer length is set to 322 twelve-bit words, by the program.

When a new position select is received, step three above, the unit verifies that the correct track is found before a "ready" status is established. Therefore, a more elaborate sequence *must* be used to verify position.

The Disk Storage Unit is an essential component in the operation of the 6600 computing system. Typically, one PPU is assigned to transfer data as needed between Disk Storage and Central Storage. Access time to the correct position is probably the most serious throughput limitation in using this unit. Although latency can also be a factor, transfer of more than one sector of 64 central storage words is preferred if possible. This, of course, depends on the nature of data files being stored.

SYSTEMS OPERATION

VIII

In order to function at all the Control Data 6600 requires an operating system. During development of the computer, an experimental operating system was also developed. This was called the Chippewa Operating System, referring to the Chippewa Laboratory of Control Data. This system has formed the nucleus of later operating systems for the 6600. Some of the interesting features of the Chippewa Operating System are discussed in this Chapter. It is not within the scope of this book to give a complete exposition of the 6600 operating system.

It should be noted that the operating system described here is one of many that might be conceived for the 6600. Other systems may attempt different emphasis on the handling of jobs and resources.

A. FILES

Information, both programs and data, may enter and leave the computing system through the use of files. For illustration, the punched card reader, the disk storage unit, and the printer are of interest.

An INPUT FILE is established on the disk storage unit from the punched cards being read. The PPU's are very conveniently used for this purpose.

Similarly, on completion of a job an OUTPUT FILE is established on the disk storage unit. A disposition is assigned to this output file, such as PRINT, PUNCH, PUNCH BINARY, and so on.

A collection of input files can form an input queue just as a collection of output files can form an output queue for printing, punching, and so on.

It is convenient, if not essential, that a common set of definitions be used in such files in order that they may be transferred from one device to another. In the Chippewa Operating System, file names must begin with an alphabetic character and may contain up to seven alphanumeric characters. Except for a magnetic tape file which may have more than one file mark, files consist of a single physical file divided into logical records. A *logical record* consists of a number of 60-bit words containing either coded or binary data. The form of storage and the method of separating logical records depend on the equipment.

This definition of logical records makes it possible to have equivalent forms of file on several devices, taking advantage of each form of storage.

Card Files use a format as follows, which allows the use of both binary and coded cards. Holes are punched in column 1 in the rows listed.

• End of logical record	Rows 7, 8, and 9
• End of file	Rows 6, 7, 8, and 9
• Binary card	Rows 7 and 9
• Coded card	Rows 7 or 9, not both

This allows up to fifteen central storage words on a binary card, starting at column 3. For binary cards, a word count is included in column 1 and a checksum in column 2. Column 80 includes a binary serial number.

Coded cards are translated on input from Hollerith code to display code and packed 10 columns, or characters, to a central word.

Disk Files make use of the efficient storage packing of the disk storage unit. The alternate sectors of the Disk Storage make up a half-track. Storage for a disk file is reserved by the monitor in half-tracks as needed. Each disk file must start at the first sector of a half-track. When a half-track is full, the file is continued at the first sector of another half-track.

Two control bytes are recorded at the beginning of each sector. The first provides linkage data to the next sector. If the file is continued on the same half-track, this first byte contains a sector number. If the file is continued on a new half-track, this first byte contains a logical half-track number. If no further information exists in the file, this first byte is zero. The second control byte specifies the number of central storage words of data in the sector. End-of-logical record is indicated if this number is less than 64. Both control bytes are zero for end-of-file.

The operations which may be performed on a file include:

- Read—coded or binary,
- Write—coded or binary,
- Backspace,
- Write end record,
- Write end file mark.

B. TABLES

All PPU requests for input-output involve a set of tables defining the nature of the file and where it is stored.

A central program may call for an I/O operation by a simple message left in its program space. This message is scanned by a PPU acting as system monitor (Section D, this Chapter). The message includes the program name and other pertinent information.

A portion of Central Storage is utilized to maintain tables and communication areas for system control purposes. This is called *central resident.*

The central resident contains an *equipment status table* EST which contains an entry for every equipment connected to peripheral channels. Each entry contains:

- Address of control point (Section D, this Chapter) to which this equipment is currently assigned,
- Channel number to which equipment is attached,
- Equipment synchronizer and unit number,
- Equipment type code; such as, tape, disk, etc., and
- A ready bit.

The central resident also contains a *channel status table* CST which relates the current assignment of PPU's to channels.

The name and status of all files are stored in the central resident area as well. Two central storage words are used for each file in the table, the first word belonging to the *file name table* FNT and the second word belonging to the *file status table* FST.

The first word entry, FNT, contains:

- File name of up to seven alphanumeric characters, starting with a letter, stored in display code.
- File type, may be one of four types.
  0 INPUT file—stored form on disk
  1 OUTPUT file—stored form on disk
  2 COMMON file—may be passed from job to job
  3 LOCAL file—discarded at end of job
- Priority.
- Control point number (Section D, this Chapter).

The second word entry, FST, contains a pointer to the equipment status table EST identifying the device and other data as follows.

DISK	Equipment number, pointer to EST
	First track of file
	Current track of file
	Current sector
	Last buffer status interlock

TAPE	Equipment number, pointer to EST Last block number Last buffer status interlock
CARD	Equipment number, pointer to EST Card count in record End of job flag Last buffer status interlock
PRINTER	Equipment number, pointer to EST Last buffer status interlock

The above tables provide very flexible system control over I/O operations since equipment can be allocated symbolically. An operation on a named file is performed when the file name, location of central storage buffers, and a code for the operation are specified. A PPU can look up the name in the file name table FNT and the equipment number from the file status table FST and perform the requested operation. When the operation is complete, the number one is added to the operation code, which is initially even, and entered into the last buffer status area of FST. This serves as an interlock so that only one PPU at a time uses the file.

C. CIRCULAR BUFFER FOR I/O

For transferring files between I/O and central storage, a PPU may call for a circular buffer program labeled CIO. The user central program specifies a file name and operation code, plus information about the circular buffer in central storage; then CIO performs the operation.

Before the central program calls for CIO, five central storage words are prepared as follows.

	Word		Remarks
WORD 1	File Name	Op-code	Name
2	–	FIRST	Beginning address
3	–	IN	Current input address
4	–	OUT	Current output address
5	–	LIMIT	Last address + 1

The circular buffer and the buffer parameter area (above) must be within the field length of the job, and addresses are relative to the job reference address RA.

A central program can then call on CIO by entering in its message area, location RA + 1, the code CIO and a pointer to the buffer parameters.

System monitor detects this CPU message and finds a free PPU to perform the task, then clears RA + 1 to signal the CPU that the circular buffer is begun.

The processing flow is shown for CIO in Figure 92.

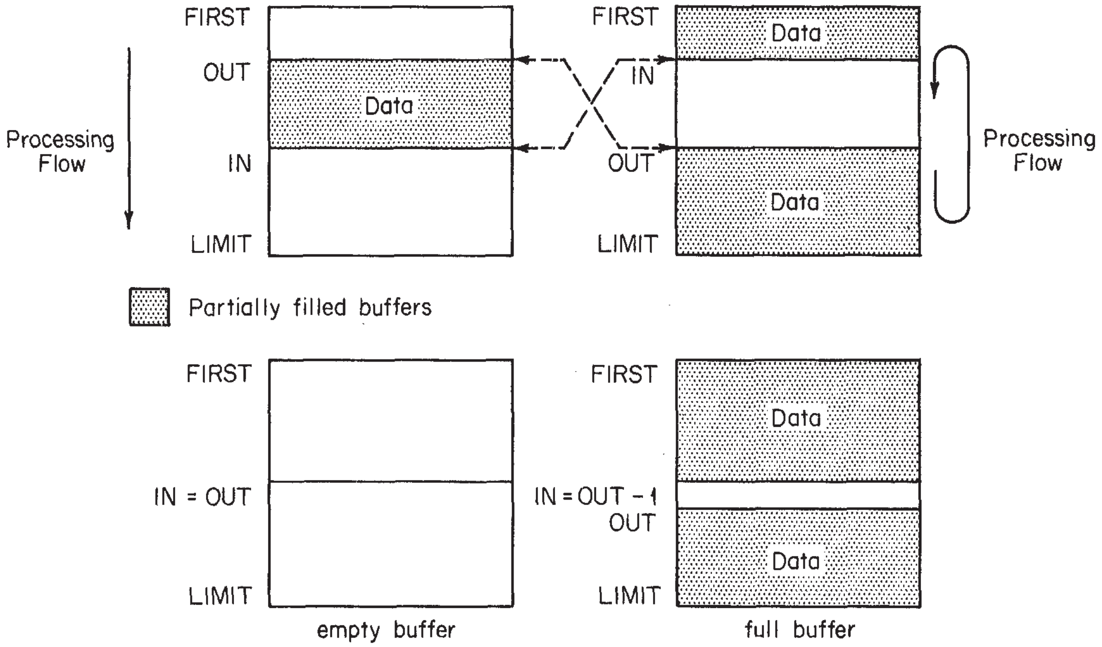

FIRST
OUT
Data
IN
LIMIT
Processing Flow
FIRST
Data
IN
OUT
Data
LIMIT
Processing Flow
Partially filled buffers
FIRST
IN = OUT
LIMIT
empty buffer
FIRST
Data
IN = OUT − 1
OUT
Data
LIMIT
full buffer

FIGURE 92 Circular buffer I/O (CIO) processing flow.

The circular buffer is used in either direction. The PPU may load the buffer and the CPU empty it, or the CPU may load and the PPU empty it. As far as the buffer is concerned, though, OUT defines the address for extraction of data from the buffer, and IN defines the address for entry of new data. As data is extracted, OUT is stepped around the buffer but never beyond IN.

Since the buffer parameters are located within the job space, the central program can step along with the PPU as long as the buffer is not exceeded or as long as OUT never exceeds IN.

D. JOB PROCESSING

A job is made up of one or more CPU programs which are executed with data files. Jobs are processed in three sequential, but independent, stages:

- Input
- Execution
- Output

The multi-programming nature of the operating system allows many jobs in the *input* or *output* stages of processing. Seven jobs may be in the *execution* phase and are handled by *control points*. Figure 93 illustrates the system elements in use in this "three phase" job processing.

The system reads an entire job from the card reader, using one of the "pool" PPU's and stores it as an *input file* on the system disk. Many jobs may be entered in this manner to form an input queue. Typically, an input

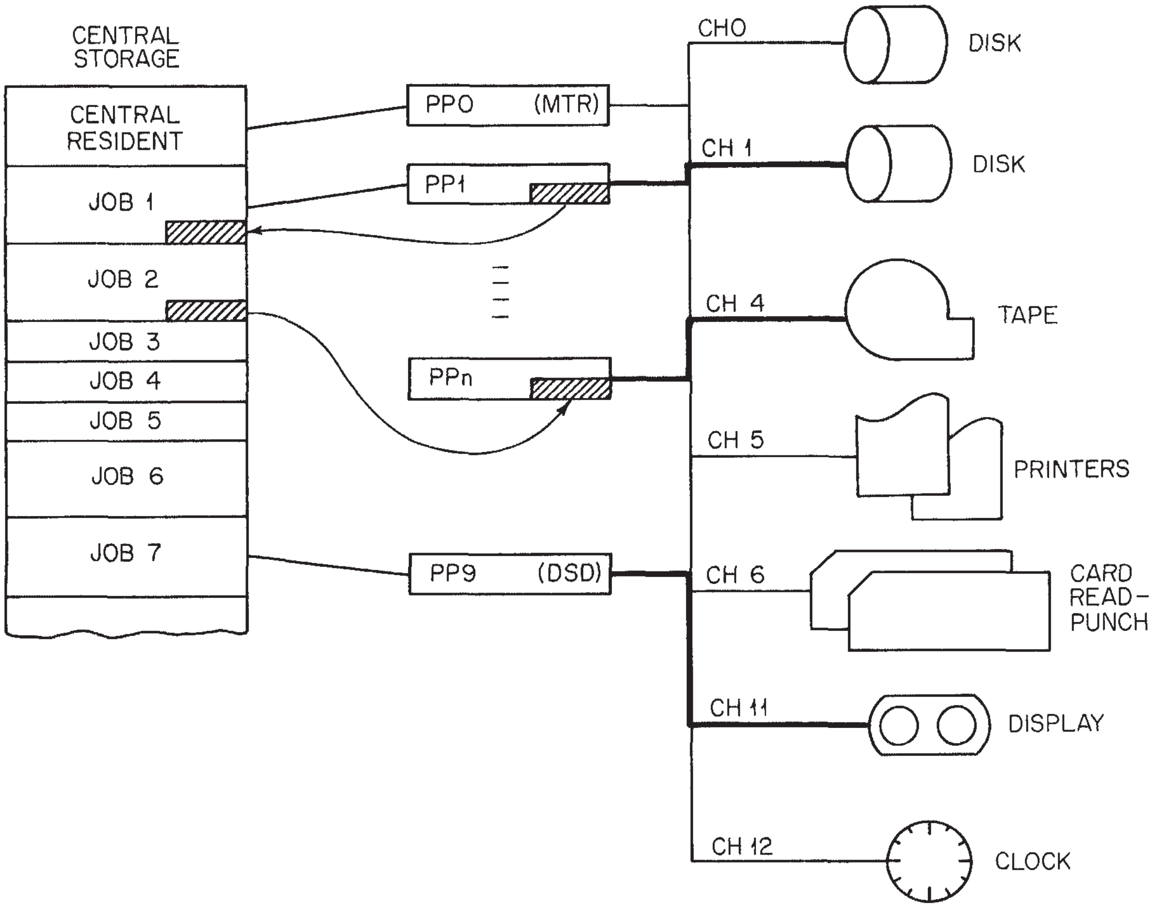

CENTRAL STORAGE
CENTRAL RESIDENT
JOB 1
JOB 2
JOB 3
JOB 4
JOB 5
JOB 6
JOB 7
PP0 (MTR)
PP1
PPn
PP9 (DSD)
CH0
CH 1
CH 4
CH 5
CH 6
CH 11
CH 12
DISK
DISK
TAPE
PRINTERS
CARD READ-PUNCH
DISPLAY
CLOCK

FIGURE 93

file is made up of three logical records; control cards, program cards, and data cards.

The system executes a job independently of the job input step, by bringing the job to a *control point.* Once in a control point, the job proceeds by following the directives of the control cards (in the input file in disk storage). During execution, the system accumulates output data on the system disk.

When the system has completed or has processed the last control card for a job, it changes the file of accumulated output data to an *output file.* A disposition, such as PRINT, PUNCH, etc., is assigned. Output files produced in this manner make up an output queue.

E. SYSTEM MONITOR MTR

The operating system functions under the overall direction of the system monitor program MTR, located in PPO. This program repeatedly scans the communication linkages in the central resident area for requests for monitor action from the CPU or from PPU's.

MTR is used in the assignment and release of all PPU's data channels, disk storage, and other I/O equipment.

Communication between MTR and the PPU's is accomplished through ten PPU communication areas in central storage. Each communication area contains:

- Word 0 —processor input,
- Word 1 —processor output,
- Words 2–7—message buffer.

A PPU idles in its resident program as long as word 0 is cleared. MTR enters a control word in word 0 of the selected PPU communication area in order to call a transient PPU program to that PPU. The resident peripheral program of the selected PPU senses the processor input entry in word 0, locates the called program, and loads it into PPU storage.

After loading, the PPU resident program then jumps to the beginning of the transient program. Following completion of the transient program, word 0 is cleared.

A PPU may communicate a request to MTR by entering a value in its word 1. A request too long for a single location is continued in the message buffer. MTR repeatedly scans the communication area; when a message is found, MTR jumps to a subroutine to process the request, then continues scanning.

F. CONTROL POINTS

As many as seven jobs may be active in central storage at one time. Each active job is assigned to a *control point* area which contains all information necessary to control the job and to resume operation after interrupt.

A job is brought to a control point by the system monitor MTR in order to begin the execution phase. Each control point contains data as shown in Figure 94 using 200 (octal) central *storage* addresses. See page 170.

A number of conditions are possible for a job in a control point, such as:

A	Active
B-G	Waiting for execution
X	Waiting recall
Blank	No requirement for CPU

Conditions A through G represent a queue of central jobs with A in execution and the rest waiting.

Condition X, waiting recall, arises by an explicit action of the job while it is being executed in the CPU. This action communicates directly to the monitor to temporarily relinquish control. This may be used, for example, to buffer I/O. By examining a communication word in central storage, the central program can determine the progress of input-output. If a point is reached where further progress is temporarily impossible, the central program may halt and activate the recall condition.

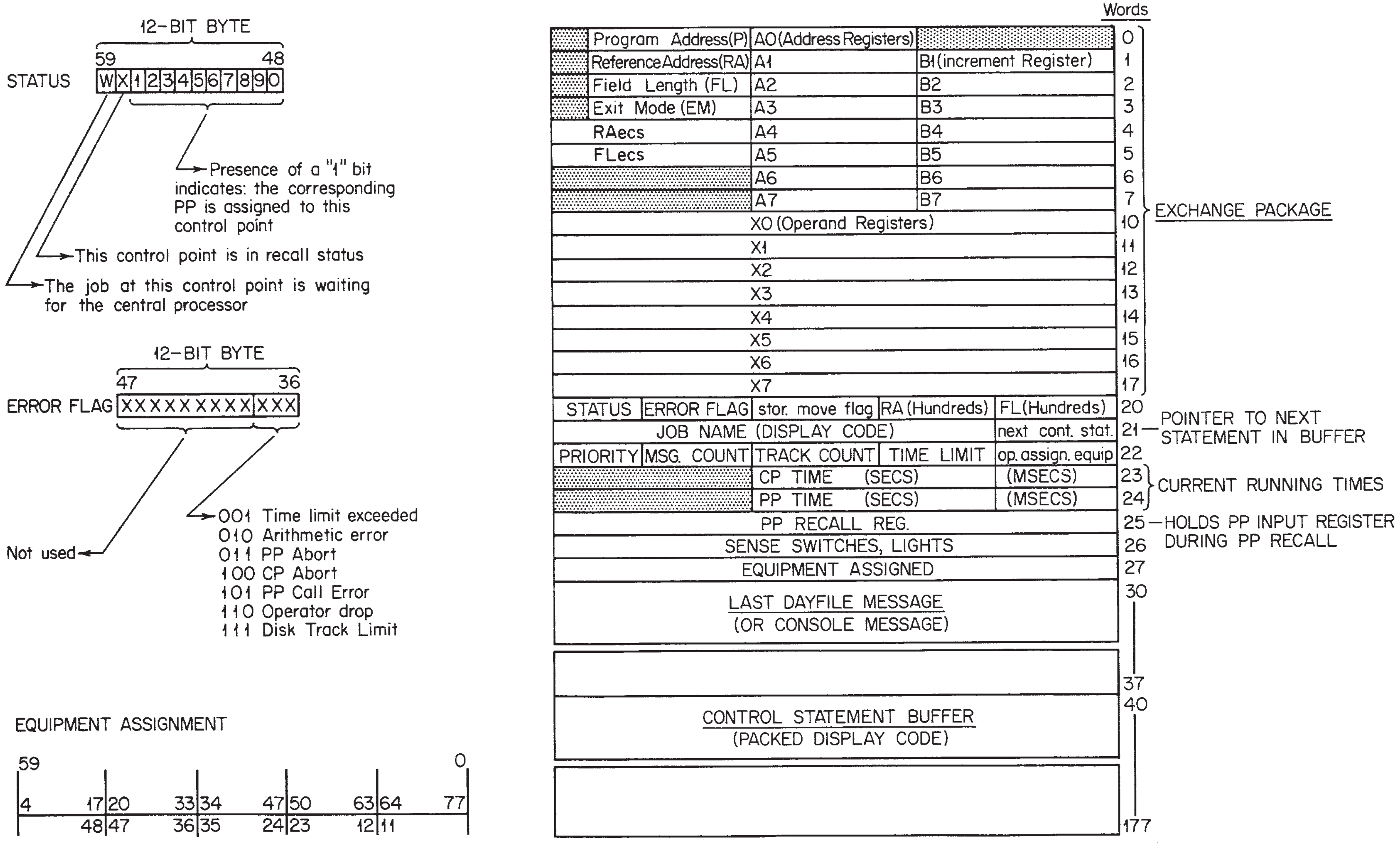

12-BIT BYTE
59
48
STATUS
W X 1 2 3 4 5 6 7 8 9 0
Presence of a "1" bit indicates: the corresponding PP is assigned to this control point
This control point is in recall status
The job at this control point is waiting for the central processor
12-BIT BYTE
47
36
ERROR FLAG
XXXXXXXXX XXX
Not used
001 Time limit exceeded
010 Arithmetic error
011 PP Abort
100 CP Abort
101 PP Call Error
110 Operator drop
111 Disk Track Limit
EQUIPMENT ASSIGNMENT
59
0
4
17 20
33 34
47 50
63 64
77
48 47
36 35
24 23
12 11
Words
Program Address(P)
A0(Address Registers)
Reference Address(RA)
A1
B1(increment Register)
Field Length (FL)
A2
B2
Exit Mode (EM)
A3
B3
RAecs
A4
B4
FLecs
A5
B5
A6
B6
A7
B7
X0 (Operand Registers)
X1
X2
X3
X4
X5
X6
X7
STATUS
ERROR FLAG
stor. move flag
RA (Hundreds)
FL (Hundreds)
JOB NAME (DISPLAY CODE)
next cont. stat.
PRIORITY
MSG. COUNT
TRACK COUNT
TIME LIMIT
op. assign. equip
CP TIME (SECS)
(MSECS)
PP TIME (SECS)
(MSECS)
PP RECALL REG.
SENSE SWITCHES, LIGHTS
EQUIPMENT ASSIGNED
LAST DAYFILE MESSAGE
(OR CONSOLE MESSAGE)
CONTROL STATEMENT BUFFER
(PACKED DISPLAY CODE)
0
1
2
3
4
5
6
7
10
11
12
13
14
15
16
17
20
21
22
23
24
25
26
27
30
37
40
177
EXCHANGE PACKAGE
POINTER TO NEXT STATEMENT IN BUFFER
CURRENT RUNNING TIMES
HOLDS PP INPUT REGISTER DURING PP RECALL

FIGURE 94 Control point areas and exchange jump area.

When a PPU has completed the I/O task or after a fixed time, the monitor is alerted to recall the control point. The control point is converted from X status and a search is made for control point priority to determine if the control point should be entered into the stack.

Storage is allocated to jobs at control points so that the order is the same as the control points themselves. Moves of storage are made after completion of a job only as needed to accommodate to the requested space of a new job.

A PPU attached to a control point can request or release storage via the monitor program MTR. This commonly takes place when a new job is brought to the control point with a different requirement than the previous job occupying that control point. The PPU specifies the space required, as determined from the job control cards.

For the actual movement of storage, where needed, the CPU is called on to perform a storage move program from the system library.

G. SUMMARY

There are, of course, many other interesting details of the Chippewa Operating System and of the systems which evolved from it. The methods used are intimately related to the structure and organization of the 6600 Computer. The short discussion in this chapter should give some insight into the use of PPU's for system control over input output. A single monitor PPU with a number of "worker" PPU's can perform with considerable flexibility while maintaining simplicity and discipline.

A key hardware feature is the CPU Exchange Jump which provides very rapid and simple interruption of one job and transfer to a new job. Another hardware feature, which affects and determines system strategy, is the high rate of transfer within storage, and particularly between central storage and extended core storage.

For brevity, no discussion was given here to assembly language or higher level language, such as FORTRAN. This operating system is structured to facilitate such language usage as well as to perform on-line diagnostic and maintenance programs.

It should also be obvious that many other considerations could be discussed. Some of these involve:

- Extended core storage.
- Central Processor monitor.
- Multiple systems.
- Overlays and segmentation.

APPENDIX

6600 TIMING NOTES

1. The times given for the CPU are computational times—the time needed after the execution start until the result is computed and stored in the result register. Times are given in minor cycles (1 minor cycle = 100 nanoseconds).
2. A functional unit cannot be reused until one minor cycle after any execution. (Result is stored by Entry Control during the minor cycle after release.)
3. A result register value may be used as an operand to another instruction as soon as the result has been stored into the register (same minor cycle). This result register will not be freed for use as a result register of another instruction until one cycle after the result has been stored into that register. (No trunk priority is considered.)
4. Instructions are issued to the functional units if:
   a. The word containing the instruction is in the stack,
   b. The functional unit(s) needed are free, and
   c. The result register(s) needed are free.

   If these conditions are not met, all further instruction issues are held until they are satisfied. Each issued 15-bit instruction requires one minor cycle before the next instruction is available for issue. Each issued 30-bit instruction requires two minor cycles before the next instruction is available for issue.
5. Execution within a functional unit does not start until the operand(s) are available. The two operands required are fetched from the registers at the same time (one operand is not loaded while the unit waits for the second operand).
6. In instructions 02–07, where more than one functional unit is used, the instruction is not issued until both functional units involved are free.
7. Times given for instructions 01–07 and 50–57 do not consider any memory conflict conditions. A practical average increase in time due to conflict may be taken as under ten percent.

8. In instructions 50–57, if i = 1, 2 . . . 5 (load from central storage instructions), the Xi register value is not available until 8 minor cycles after the start of the instruction execution (assuming no memory conflicts). When two load instructions begin execution one minor cycle apart, at least one extra minor cycle is required for execution of the later instruction. Therefore, the second executed instruction would require 9 cycles for the load, 4 cycles for the increment unit result to the A register.

9. In instructions 50–57, if i = 6 or 7 (store to central storage instructions), the Xi register is not available for a result register until 8 minor cycles after the instruction begins execution (assuming no memory conflicts). When two store instructions begin execution one minor cycle apart, one extra cycle is required for execution of the later instructions. Therefore, the second executed instruction would require 9 cycles for the store, 4 cycles for the increment unit result to the A register. A store instruction checks the X register before being issued. The X register is available as an entry operand register while the store is taking place.

10. When executing sequential instructions that are not in the stack, the minimum time is one word of instructions every 8 cycles. The time of issue of the last parcel of an instruction word to the time of issue of the first parcel of the next instruction word (while executing sequential instructions that are not in the stack) requires a minimum of 4 cycles. If the last instruction in a word is a 30-bit instruction, a minimum of 5 cycles is required from the time of issue of this instruction to the time of issue of the first instruction of the next word.

11. All 03 branches made within the stack require 9 minor cycles. An 03 branch to the next sequential word is recognized as a branch within the stack and requires 9 minor cycles.

12. When a branch out of the stack is taken, 15 minor cycles are normally required for an 03ijk instruction and 14 minor cycles for other branch instructions (considering no memory conflicts), timed from the start of the branch instruction execution to the availability of the branched-to word instruction to a functional unit (instruction ready for issue).

13. Eleven cycles are required for the 03ijk instructions when the branch is not taken (time from branch execution to issue of the next instruction) if in the stack or if falling through to an instruction within the same word. Out of stack fall-through to the next word takes 14 cycles.

14. Ten cycles are required for 04ijk — 07ijk instructions when the branch is not taken (time from branch execution to issue of the next instruction) if in the stack or falling through to an instruction within the same word. Out of stack fall-through to the next word takes 13 cycles.

15. Neither increment unit may be involved in a load operation if a store operation is to be issued, and neither increment unit may be involved in a store operation if a load operation is to be issued. The sequential loading of instruction words does not affect the load/store conditions of the increment units.

16. The operand registers are available to more than one functional unit in the same minor cycles if the units are in different groups.

GROUP 1	GROUP 2	GROUP 3
Divide	Add	Increment 1
Multiply 1	Shift	Increment 2
Multiply 2	Long Add	
Boolean		

17. The time needed for a functional unit to operate on indefinite, out-of-range or zero values is the same as for normal, in-range values (i.e., no gain or loss in execution time due to a unit recognizing an indefinite operand and setting an indefinite result).

18. An index jump instruction (02) will always destroy the stack. If an unconditional jump backward in the stack is desired, an 0400k instruction should be used (to save memory access time for instructions).

19. A return jump instruction (01) will always destroy the stack.

20. Functional unit times given on the end papers for CPU timing are measured in minor cycles of 100-nanosecond duration each.

21. Instruction times given on the end papers for PPU timing are measured in major cycles of 1000-nanosecond duration each.

INDEX

A
Accept Bus, 48–49
adder entry: make up, 85–88
adder network, 85–88; output, 88
additive adder, 64, 93
additive merge tree, 90–91, 97–98
Add Unit, 77–88; adder network, 85–88; double precision result, 81; execution time, 77; exponent calculation, 82–84; instructions executed in, 77; right shift network, 84–85; single precision result, 81
air cooling, 4
alignment shift, 84. *See also* right shift network
AND, 23–24

B
back panel wiring, 5
banks, 16; in Central Storage, 39
barrel, 141–143
basic circuit properties, 19–36; DCTL (Direct-Coupled Transistor Logic) logic circuits, 21–24; logic symbols, 24–28; packaging, 32–36; silicon transistors, 19–21; transmission lines, 28–31
block transfer, 54
Boolean functions, 60
Boolean unit, 59–63; execution time, 60; bootstrap, 157
borrow, 64–66
borrow generation, 66–67; in Add Unit, 83, 86–88
borrow pass, 66–67; in Add Unit, 83, 86–88
Branch Unit, 111–114; instructions executed in, 111–112; instruction stack, 112; partner units, 112; Return Jump, 113–114
buffer register: and instruction stack, 120, 121
building block approach, problems of, 5
busy flag, 128–130

C
cables, 32
carry. *See* borrow
carry-save network, 91, 92; scheme, 93–97
Central Processor (CPU), 9–10, 11–13
Central Processor control, 117–140; Exchange Jump, 117–120; instruction fetch, 120–123; instruction issue, 123–125; register entry/exit control, 134–136; Scoreboard, 125–134; summary, 137–140
Central Processor functional units, 57–116; Add Unit, 77–88; Boolean Unit, 59–63; Branch Unit, 111–114; Data Trunks, 69–71; Divide Unit, 101–105; ECS Coupler-Controller, 114–116; Fixed Add Unit, 63–69; Increment Units, 105–111; Multiply Unit, 88–101; names of, 57–58; Shift Unit, 71–77
Central Storage, 15–17; design considerations, 15; properties, 16
Central Storage banks, 39
Central Storage: CPU references, 110–111
Central Storage cycle, 43–44
Central Storage: ECS (Extended Core Storage) transfers, 114–116
Central Storage System, 37–56; ECS, 53–55; ECS Coupler-Controller, 55–56; general techniques, 37; inter-

leaved storage, 44–47; storage bus system, 51–53; storage module, 37–44; Stunt Box, 47–51
Central Storage: read/write pyramids, 154
chassis, 34–36
chassis interconnection, 29
chassis: Central Storage, 38–39
Chippewa Laboratory, 163
Chippewa Operating System, 164
circuit packaging, 33
circular buffer program, 166–167
clear/set network: and flip-flop, 26, 27
clock oscillator, 32
coaxial transmission circuit, 29, 30
coaxial cable connections, 30–31
coefficient, 77–78
coincident current, 37, 39–40
complex module approach, 6
computer, motivation for, 1–4
computer: justification for large, 4
conflicts, 125–127
Control and Peripheral Processors, properties of, 10
control system, 7
Courant Institute, 3

D
data control, entry and exit, 134–135
data transfer, 14
Data Trunks, 69–71; priorities in, 71
DEAD START, 11, 154–157; program for, 156–157
designators, 127–128
destructive readout storage (DRO), 41, 42
Direct-Coupled Transistor Logic circuit (DCTL), 21–24; ground rules for use, 25
disk half-track, 160
disk sector, 159
disk storage, 5, 157–161; control by PPU, 160–161
disk track, 159
Divide Unit, 101–105; execution time, 101; exponent calculation, 104–105; instructions executed in, 101; rounding in, 105
double precision: add, 81; multiply, 88–89

E
end-around carry, 64
end-around borrow, 83–84, 104. *See also* end-around carry
epitaxy, 20–21
error mode, 118
example programs, 137–140
Exchange Jump, 11, 116–120; and CPU, 118–119; effect in PPU, 118; effect on ECS transfer, 116; execution time, 118; interrupt, 55; usefulness, 120
Exchange Jump interrupt, 55
exchange package, 118
exponent, 77–78; bias, 78
exponent calculation: in add, 82–84; in divide, 104–105; in multiply, 98–99
Extended Core Storage (ECS), 17, 18, 53, 54; Coupler-Controller, 17–18, 114–116; organization, 17–18; properties, 17; storage hierarchy, 53
ECS Coupler-Controller, 114–116; instructions executed in, 114; interrupts, 116; timing, 115–116
ECS timing, 54–55
ECS word length, 54

F
fall-through, 125
fan-in. *See* loading
fan-out. *See* loading
Fernbach, S., 2–3
ferrite magnetic cores, 4, 37, 39
Field Length, 50, 118; ECS, 53, 56
Fixed Add Unit, 63–69; block diagram, 67–68; instructions executed in, 68; partner to Branch Unit, 68–69
fixed-point numbers, 78
flip-flop, 26–28
floating point numbers, 78
floating point: addition and subtraction. *See* Add Unit; divide. *See* Divide Unit; format, 77–78; multiply. *See* Multiply Unit; nonstandard values, 78–79; normalize. *See* Shift Unit; scaling instructions, 72, 77–78
Freon cooling, 5, 34–35
function translation, 122
functional parallelism, 1, 5, 6, 12, 57, 58

functional units, 6, 12–13
functional overlap, 7

G
germanium transistors, 4, 19

H
Harrison, M. C., 3–4
hopper, 48–50. *See also* Stunt Box

I
inching, 121
increment addition, 111
increment functional units, instructions for, 106
Increment Units, 105–111; addition in, 111; as partner to Branch Unit; instructions executed in, 106–110; storage references, 110–111
indefinite value, 78–79; in multiply, 99
indexing operations, 105–106
infinite value, 78–79; in multiply, 99
input-output files, 165
input-output tables, 165
in-stack branches, 112–113
instruction fetch, 14, 112–113, 121
instruction flexibility, 7
instruction formats, 58–59; peripheral processor, 143
instruction issue, 122; restrictions, 124
instructions: in Central Processor, 13
instruction stack, 13; and buffer register, 120–121; and time penalty, 123
instruction words, 112–113
interleaving in Central Storage, 5
interrupt (Exchange Jump), 114, 118, 120
inverter, 21–22

J
job processing, sequential stages of, 167
jump. *See* Branch Unit

L
large computers, 1–4
linear-select, 54
logic circuit, 5, 21; and truth tables, 23; configuration of, 23; construction, 6; design constraints, 28; diagram of, 24, 25
logical functions, 60
logical inversion (NOT), 22
loading, 25
look-behind, 123
loops, 13, 113

M
magnetic core storage, 37
magnetic cores, two-dimensional array of, 40, 41
magnetic properties of ferrite core, 39–40
magnetic tape, 4
Major Cycle, 16
memory. *See* storage *references.*
micro instructions, 7
Minor cycle, 16, 31, 71
modules, 32–36
monitor state, 119
motor-alternator, 35
MTR (system monitor program): use in PPU's, 168–169
multiple processors, 18
multiprocessing, 1; advantages of, 6; conditions permitting, 10; theory of, 6
multiprogramming, 3, 5, 7–8
Multiply Unit, 88–101; carry-save network, 93–97; execution time, 88; exponent calculation, 98–99; instructions executed in, 88; merge network, 97–98; methods, 89–92; rounding, 99–101; sequence of operations, 92–93

N
negative zero, 64
normalize, 71
normalize network, 75–77
normalized arithmetic, 89
normalization in addition, 81; in multiply, 89, 99
NOT, 23–24
NPN transistor, 19–20

O
one's complement, 63
OR, 23–24

organization of 6600, 9–1
output file: on disk storage unit, 163
overflow, 78–79, 81; in multiply, 99

P

pack, 71
packaging, 5–6, 32–36
paging, 3
parallel addition, 63–66
parallel functional units, 9
parallel shift network, 73–75, 85
parcels: in instruction word, 121
partial product, 89–90
Peripheral and Control Processors: properties, 10
peripheral channels, 10–12
peripheral instructions: jumps, 143–144; Central Processor, 148–149; Central Storage, 149–150; direct-indirect-index, 146–148; input/output, 150–152; no address, 144–146
peripheral processors, 5, 7, 10–12
Peripheral Processor Units (PPU), 7, 9; access to Central Storage, 149–150; barrel design, 141–142; communication with, 11; data flow between, 150–152; effect of Exchange Jump, 118; instructions: direct address mode, 146; index mode, 147; indirect address mode, 147; jump, 143 processing requirements, 141; Read pyramid, 154; Write pyramid, 154
PPU barrel, registers in, 152; register operations of, 153
peripheral subsystem, 10–12, 141–161; Dead Start, 154–157; peripheral processors, 141–154
population count, 101, 105
PNP transistor, 20
pre-rounding, 101
primary storage, 45–46
priority network: Central Storage, 50–51
priority network. *See* Stunt Box
program address register, 13
protection, 53
pseudo-carry, 93
pseudo-sum, 93

R

read flag, 130–134
Read Pyramid: in PPU, 154
real-time clock, 152, 156
re-entrant code, 114
referencing Central Storage. *See* Stunt Box
registers, 59; in Central Processor, 13
relative address, 50; and ECS, 53, 56
release signal, 132, 134
relocation, 53
reservation control. *See* Scoreboard
reservations, 128–130
Return Jump instruction, 113–114
right shift network, method of, 84, 85
rounding, 76; in add, 84; in divide, 105; in multiply, 99–101

S

Schwartz, J. T., 3–4
Scoreboard, 14, 125–134; conflicts resolved by, 125; design, 125; operations of, 128–134
scratch pad, 7
secondary storage: in storage hierarchy, 44, 45
secondary storage unit, 157–159
segmentation, 3
serial adder, 64
shift, 71
shift apparatus: operation principle, 73
shift logic, 73–75
Shift Unit, 71–77; instructions executed in, 71–72; number of modules in, 75
silicon, advantages of, 19
silicon planar transistor, 19–21
silicon transistors, 5, 6
6500 computer, 120
6600 organization, 9–18; Central Processor, 12–15; Central Storage, 15–17; Extended Core Storage, 17–18; peripheral subsystem, 10–12
6636 disk, 157–161; characteristics of, 160; control sequence for, 160–161
6638 Magnetic Disk Storage. *See* secondary storage unit
SKIP command, 124

slot time-sharing, 142–143, 152–154
storage address bus, 48, 51
storage hierarchy, 44–45
storage module, properties of, 37–38
storage, operation of, 39
storage protection, 16–17
storage references, 43, 44
storage bus system, 51–53
Stunt Box, 15, 47–51; hopper, 48–49; priority network, 50; referencing Central Storage, 52; storage bus system, 51-53; tag generator, 51
subtractive adder, 64
super-word. *See* sword
swapping, 17
switching time, 22, 25–26; Central Storage, 40
sword, 54, 115
synchronous overlap, 47
synchronous storage, 47
system library, 171
system monitor program (MTR): use in PPU's, 168–169
systems operation, 163–171; circular I/O buffer, 166–167; control points, 169–171; files, 163–164; job processor, 167–168; system monitor MTR, 168–169; tables, 165–166

T
tag generator: Central Storage, 51. *See also* Stunt Box
test points, 33–34
time penalty, in Central Storage, 123
time-sharing, 3
time slicing, 120
timing in Central Storage, 50–51
timing for example programs, 137-140
transfer. *See* Branch unit
transmission lines, 28–31
transistor, 19–22, 32; characteristics of, 21, 22; current, 19; power, 19, 32; temperature, 19, 32; voltage, 22
twisted pair connections, 28–30

U
underflow, 78–79, 81–82; in multiply, 99
Unit and Register Reservation Control. *See* Scoreboard
unpack, 71

V
voltage levels, 22–23

W
wire lengths, 6
wiring: within modules, 28
Write Distributer, 52
Write pyramid: in PPU, 154
word, 115
Worlton, W. J., 2, 3

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 78 77 76 75 74 73 72 71 70

PERIPHERAL AND CONTROL PROCESSOR INSTRUCTION EXECUTION TIMES

OCTAL CODE	NAME	TIME (MAJOR CYCLES)
00	Pass	1
01	Long jump to m + (d)	2-3
02	Return jump to m + (d)	3-4
03	Unconditional jump d	1
04	Zero jump d	1
05	Nonzero jump d	1
06	Plus jump d	1
07	Minus jump d	1
10	Shift d	1
11	Logical difference d	1
12	Logical product d	1
13	Selective clear d	1
14	Load d	1
15	Load complement d	1
16	Add d	1
17	Subtract d	1
20	Load dm	2
21	Add dm	2
22	Logical product dm	2
23	Logical difference dm	2
24	Pass	1
25	Pass	1
26	Exchange jump	1
27	Read program address	1
30	Load (d)	2
31	Add (d)	2
32	Subtract (d)	2
33	Logical difference (d)	2
34	Store (d)	2
35	Replace add (d)	3
36	Replace add one (d)	3
37	Replace subtract one (d)	3
40	Load ((d))	3
41	Add ((d))	3
42	Subtract ((d))	3
43	Logical difference ((d))	3
44	Store ((d))	3
45	Replace add ((d))	4
46	Replace add one ((d))	4
47	Replace subtract one ((d))	4
50	Load (m + (d))	3-4
51	Add (m + (d))	3-4
52	Subtract (m + (d))	3-4
53	Logical difference (m + (d))	3-4
54	Store (m + (d))	3-4
55	Replace add (m + (d))	4-5
56	Replace add one (m + (d))	4-5
57	Replace subtract one (m + (d))	4-5
60	Central read from (A) to d	min. 6
61	Central read (d) words from (A) to m	5 plus 5/word
62	Central write to (A) from d	min. 6
63	Central write (d) words to (A) from m	5 plus 5/word
64	Jump to m if channel d active	2
65	Jump to m if channel d inactive	2
66	Jump to m if channel d full	2
67	Jump to m if channel d empty	2
70	Input to A from channel d	2
71	Input (A) words to m from channel d	4 plus 1/word
72	Output from A on channel d	2
73	Output (A) words from m on channel d	4 plus 1/word
74	Active channel d	2
75	Disconnect channel d	2
76	Function (A) on channel d	2
77	Function m on channel d	2

Date: Fri, 26 Apr 2002 12:12:54 -0400
From: Beth.Wollar@pearsoned.com
Subject: RE: permission to copy out of print book
To: uban@ubanproductions.com
X-Mailer: Internet Mail Service (5.5.2653.19)

April 26, 2002

Dear Mr. Uban:

Thank you for your email regarding DESIGN OF A COMPUTER by James E. Thornton. Rights to this title have reverted to the author. Please contact him for permission to copy or scan the book. The latest address we have for Mr. Thornton is [redacted]

Sincerely,

Beth Wollar, Supervisor
Rights and Permissions
Pearson Education
1900 East Lake Ave.
Glenview, IL 60025

-----Original Message-----
From: Tom Uban [mailto:uban@ubanproductions.com]
Sent: Wednesday, April 24, 2002 1:38 PM
To: Permissions, Glenview
Subject: permission to copy out of print book

Dear Mr. Ravas,

I am interested in obtaining permission to make copies of the following book:

Title: Design Of A Computer: The Control Data 6600
Author: J. E. Thornton
Publisher: Scott Foresman
Year: 1970
Library of Congress Catalog Number: 74-96462

This book is of a technical nature, describing a computer design which was produced in the late 1960s. It seems unlikely that it will ever be published again, however there exists a small number of enthusiasts who have an interest in the Control Data 6600 and what Mr. Thornton had to write about it. Locating a copy of this book is difficult at best.

For this reason, I am asking for permission for one (or both) of the following:

1. make a limited number of copies of the book
2. make an electronic scan of the book available to interested parties

There is no intent to profit from this request.

Thank you for your time.

Tom Uban

2 May 2002

Dear Mrs. Thornton,

I hope that my letters are not becoming an annoyance. I heard back from the permissions department at Prentice Hall, with regarc to the rights to copy the book. Here is what they had to say:

> Dear Mr. Uban:
>
> Thank you for your email regarding DESIGN OF A COMPUTER by James E. Thornton. Rights to this title have reverted to the author. Please contact him for permission to copy or scan the book. The latest address we have for Mr. Thornton is [redacted]
>
> Sincerely,
>
> Beth Wollar, Supervisor
> Rights and Permissions
> Pearson Education
> 1900 East Lake Ave.
> Glenview, IL 60025

So, it would seem that I need to ask you for this permission. Since there is a small group of people who are currently studying the CDC 6600 machine and who would each be interested in locating a copy of the book, I am requesting permission either to make a limited number of copies of the book, or to make an electronic scan of the book available to those who are interested.

Please be assured that there is no intent on my part or the part of others to profit from this request. It is solely for the purpose of preserving and disseminating the information contained therein.

With best regards,

Tom Uban

5/14/02

Mr. Uban -
Your letters are not an annoyance. Jim is pleased that there is some interest in his book and be assured there is no suspicion of profiting from it. He gladly gives permission and is glad of the possibility that the information is helpful in some measure.

Sincerely
Gloria C. Thornton
(Mrs. James)
- Jim Thornton